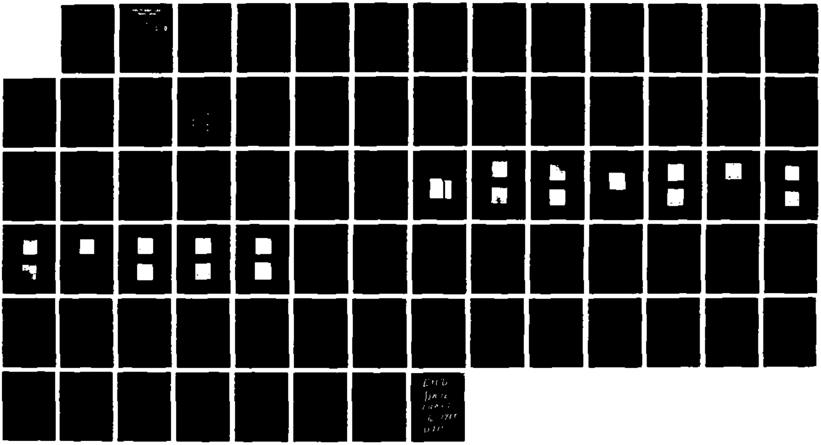
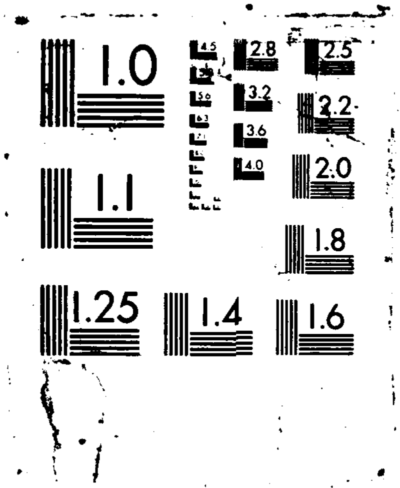
AD-A192 406 FAULT DIAGNOSIS IN DISTRIBUTED COMPUTER NETWORKS(U) 1/1
NAVAL POSTGRADUATE SCHOOL MONTEREY CA I DINCER DEC 87

UNCLASSIFIED F/G 12/8 NL




1.0
4.5
5.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

FAULT DIAGNOSIS IN DISTRIBUTED COMPUTER NETWORKS

by

Ibrahim DINCER

December 87

Thesis Advisor Jon T. Butler

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
A192 406

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
NPS62-87

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Naval Postgraduate School

6b. OFFICE SYMBOL *(If applicable)*
62

7a. NAME OF MONITORING ORGANIZATION
Naval Postgraduate School

6c. ADDRESS *(City, State, and ZIP Code)*
Monterey, CA. 93943-5000

7b. ADDRESS *(City, State, and ZIP Code)*
Monterey, CA. 93943-5000

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO

11. TITLE *(Include Security Classification)*
Fault Diagnosis In Distributed Computer Networks

12. PERSONAL AUTHOR(S)
Dincer, Ibrahim

13a. TYPE OF REPORT
Master's Thesis

13b. TIME COVERED
FROM ________ TO ______

14. DATE OF REPORT *(Year, Month, Day)*
1987, December

15. PAGE COUNT
75

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Preparata-Metze-Chien, Computer-Aided-Design

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This thesis introduces the concept of a diagnosis algorithm in the context of the Preparata-Metze-Chien (PMC) model. It represents a Computer-Aided-Design (CAD) tool for use in analyzing such algorithms. That is, with this tool, the user can establish a multiprocessor system, a set of test outcomes and then analyze the properties of specified distributed diagnosis algorithm. Examples in this thesis include a system in which: 1. Correct diagnosisis achieved in a small number of iterations. 2. Correct diagnosis is never achieved. 3. An oscillating situation exits in which, faulty processors become alternately enabled and disabled.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

22a. NAME OF RESPONSIBLE INDIVIDUAL
Prof. Jon T. Butler

22b. TELEPHONE *(Include Area Code)*
(408) 646-3299

22c. OFFICE SYMBOL
62Bu

83 APR edition may be used until exhausted
All other editions are obsolete

☆ U.S. Government Printing Office. 1986—606-24.

FAULT DIAGNOSIS IN DISTRIBUTED COMPUTER NETWORKS

by
Ibrahim Dincer
Captain, Turkish Army
B.S., War Academy, Ankara, Turkey, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1987

Author: ______________________
Ibrahim Dincer

Approved by: Jon T. Butler
Jon T. Butler, Thesis Advisor

Bruno Shubert
Bruno O. Shubert, Second Reader

Powers
John P. Powers, Chairman,
Department of Electrical and Computer Engineering

Schacher
Gordon E. Schacher
Dean of Science and Engineering

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis introduces the concept of a distributed diagnosis algorithm in the context of the Preparata-Metze-Chien (PMC) model. It represents a Computer-Aided-Design (CAD) tool for use in analyzing such algorithms. That is, with this tool, the user can establish a multiprocessor system, a set of test outcomes and then analyze the properties of a specified distributed diagnosis algorithm.

Examples in this thesis include a system in which ;

1. Correct diagnosis is achieved in a small number of iterations.

2. Correct diagnosis is never achieved.

3. An oscillating situation exists in which faulty processors become alternately enabled and disabled.

ACKNOWLEDGEMENTS

I would like to thank to my advisor Professor Jon T. Butler for his valuable assistance and patient guidance. I appreciate his great support and encouragement.

I have to express my deep respects to my government for sending me here for this education.

I also would like to thank Dr. Dana Madison for his great contribution.

Special thanks to my wife Makbule and my son Melih for their continuous support.

I. INTRODUCTION

A. NEED FOR STUDY

The advent of inexpensive microprocessor elements has made multiprocessor computing networks much more practical. This fact has led to an increasing interest in the high reliability of such networks. The prospect of ultra reliability has inspired research into the use of computers where low reliability precluded its previous use. This includes aircraft control systems, where the Federal Aeronautic Administration (**FAA**) has specified as a standard probability of failure in a 10 hour operating period of 10^{-9} [Ref. 1].

The traditional approach to computer reliability is through redundancy, where reliable outputs are the result of a vote on three or more less reliable outputs. In the theory of system diagnosis [Ref. 2], a graph is used to model a multiprocessing system where **nodes** represent the processors and **arcs** represent tests between processors. One goal of the theory is to determine what tests achieve the highest tolerance to faults. It has been shown [Ref. 3] that for the same system reliability, greater throughput can be achieved from **system diagnosis** approach than modular redundancy. Conversely, for the same throughput, a system diagnosis approach yields greater reliability [Ref. 3].

Beginning with the Preparata-Metze-Chien model, many models have been developed for system diagnosis. The best known models are [Ref. 4].

1. Preparata-Metze-Chien(PMC) model: This model was used in this research and will be explained in Chapter II. This model is represented by $\mathbf{A_p}$ in Table 1.1.

2. **Perfect Tester:** In this model, test outcomes correspond to perfect diagnosis of faulty units. In other words, if the tested unit is **faulty** (not good), no matter what the status of testing unit is (faulty or fault-free), the test outcorne will be **fail(1)**. If the tested unit is **fault-free(good)**, the test outcome will be **pass(0)** regardless of the status of the testing unit. This model is represented by A_α in Table 1.1.

3. **1-Fail safe tester**: This model never has an incorrect zero. That means that there might be incorrect fail test outcomes (e.g., when faulty unit is testing a fault-free unit the

test outcome will be 1), but there will never be any incorrect pass (0) outcome. It is represented by A_W in Table 1.1.

4. **0-Fail safe tester:** This model never has incorrect 1. That is, when a faulty unit tests another faulty unit, the test outcome will be 0. This is an incorrect **pass** outcome. However, there is no incorrect **fail** test outcome. The model is represented by A_Y in Table 1.1.

5. A_B is a model in which a faulty unit will never incorrectly diagnose another faulty unit. However, in this model a faulty unit testing a fault-free unit will produce 0 and 1 arbitrarily.

6. $A\ \mu$ is a model in which a faulty testing unit may not correctly diagnose another faulty unit. Test outcomes can be 0 and 1 arbitrarily.

7. $A\lambda$ is a model in which a faulty testing unit always diagnoses a fault-free unit incorrectly, producing **fail** test outcome. However, a faulty testing unit produces 0 and 1 arbitrarily for a faulty tested units.

8. Partial tester: In this model, there is the possibility that a fault-free testing unit cannot correctly diagnose a faulty unit. This model is examined by Simoncini and Friedman [Ref. 5]. They considered the problem where system tests may be incomplete, i.e., that is a fault-free unit may be able to detect faulty units with percentage p ($p < 100$). This model is represented by A_{pt} in Table 1.1.

9. Zero information tester: This model provides no reliable test outcomes. This model was considered by Marion L. Blount [Ref. 6]. Several different fault detection requirements can be addressed.

a. A fault-free unit can fail to diagnose another fault-free unit.

b. A fault-free unit can fail to diagnose a faulty unit.

c. A faulty unit can give a correct diagnosis of another unit (faulty or fault-free). This model is represented by A_0 in Table 1.1

	Aα	Aw	AB	AY	Aμ	Aλ	Ap	Apt	A0
aij									
0->0	0	0	0	0	0	0	0	0	X
0->**0**	1	1	1	1	1	1	1	X	X
0->0	0	1	X	0	0	1	X	X	X
0->**0**	1	1	1	0	X	X	X	X	X

0-Fault-free unit **0**-Faulty unit

Table 1.1 Different models of system diagnosis

All the models mentioned previously apply to a graph theoretic system. Analysis of such systems is typically done by hand calculation which limits the number of units. System fault configurations is limited to some small numbers as well. Thus, the analysis of such theory is difficult. Also, there is much interest in making the model more realistic. This, in fact, inspired the models described. For example, **AB** proposed to model tests among processors consisting of comparing results of computations. The goal of this thesis is to further improve the model. Specifically, it addresses the problem of reconfiguration, where there has been relatively little study so far.

B. PROBLEM ENVIRONMENT

The **fault diagnosis** problem is to determine faulty processors given the set of test outcomes. Almost all previous studies have assumed a **central diagnoser**, which collects all of the test results and identifies faulty processors from this. This assumption simplifies the problem and avoids the complexities of reliable replacement. But a central diagnoser is also a processor, which might fail. In this case, system diagnosis may not be accurate. To provide accurate system diagnosis, the central diagnoser should be ultra reliable. This will be expensive and will require extra maintenance effort. To overcome these difficulties, **distributed system diagnosis** is proposed. In the distributed systems proposed here, the hardware required to achieve reliability is simple and can be made ultra rel:able inexpensively.

II. BACKGROUND

A. PREPARATA-METZE-CHIEN (PMC) GRAPH MODEL

A multiprocessing system is composed of n processors. Each processor is called a **unit (node)** where a unit is a well-identifiable portion of the system which cannot be further decomposed for the purpose of diagnosis. Units are indicated by Ui , $0 \le i \le n\text{-}1$. These units must be powerful enough to test other individual subunits. A **test** corresponds to an **arc** between prucessors with the arrow pointing to the tested unit. Arcs are denoted by $a_{i\,j}$, where **i** is the unit number which is doing the test, and **j** is the unit number which is tested. Each test has two outcomes, **pass** and **fail;** 0's correspond to pass test outcomes and 1's correspond to fail test outcomes. Faulty processors are indicated by $\underline{X}$'s. Figure 2.1 shows a 5 processor multiprocessor system, where U2 and U3 are faulty. A test is meaningful only if the testing unit itself is fault-free; otherwise the test outcome is unreliable.

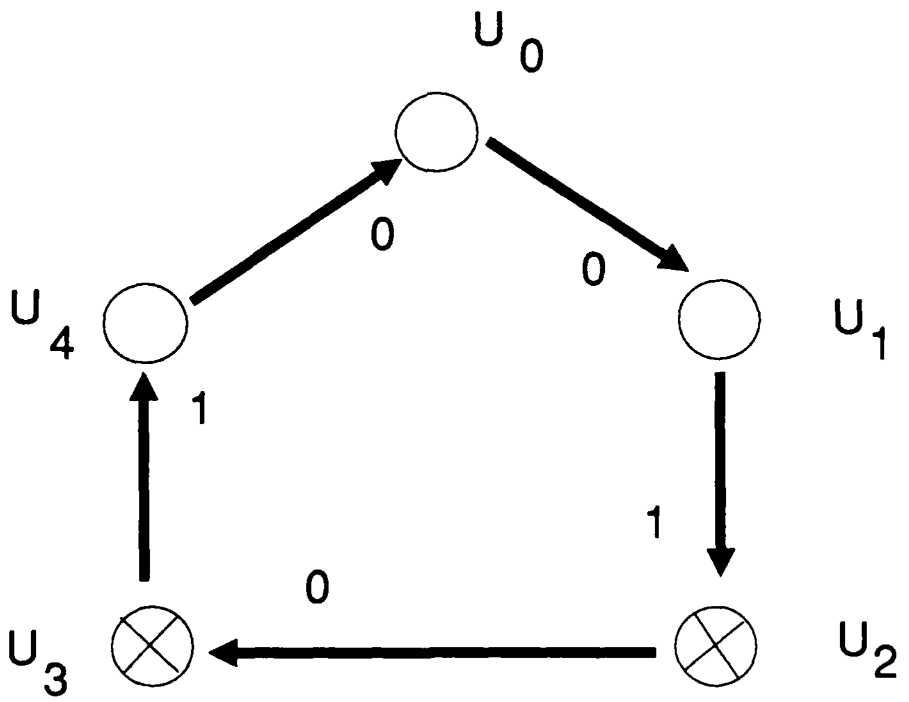


Figure 2.1 Five processors multiprocessor system with faulty units and test outcomes

Figure 2.2 shows how test results occur in the model we have chosen. The top arc goes from a fault-free node to a fault-free node and for this case a 0 (pass) outcome is always produced. The second arc goes from a fault-free node to a faulty node and for this case a 1 (fail) outcome is always produced. The third arc goes from faulty node to fault-free node and fourth arc goes from faulty node to faulty node. The outcomes of the last two cases are unpredictable and can be 0 or 1 arbitrarily.

Definition 1: The set of test outcomes **a_{ij}** represents the syndrome of the system; obviously **a_{ij}** can be assigned if and only if the corresponding testing link exists. [Ref. 3: p-848]. In Figure 2.1 the syndrome of the system for one loop will be (a_{01}, a_{12}, a_{23}, a_{34}, a_{40}) where the left to right arrangement of the **a_{ij}** is intended to reflect the direction of the loop. Diagnosis is the process of determining the faulty units given a set of test outcomes. At this point, we need to define **distinguishable** and **indistinguishable** fault patterns.

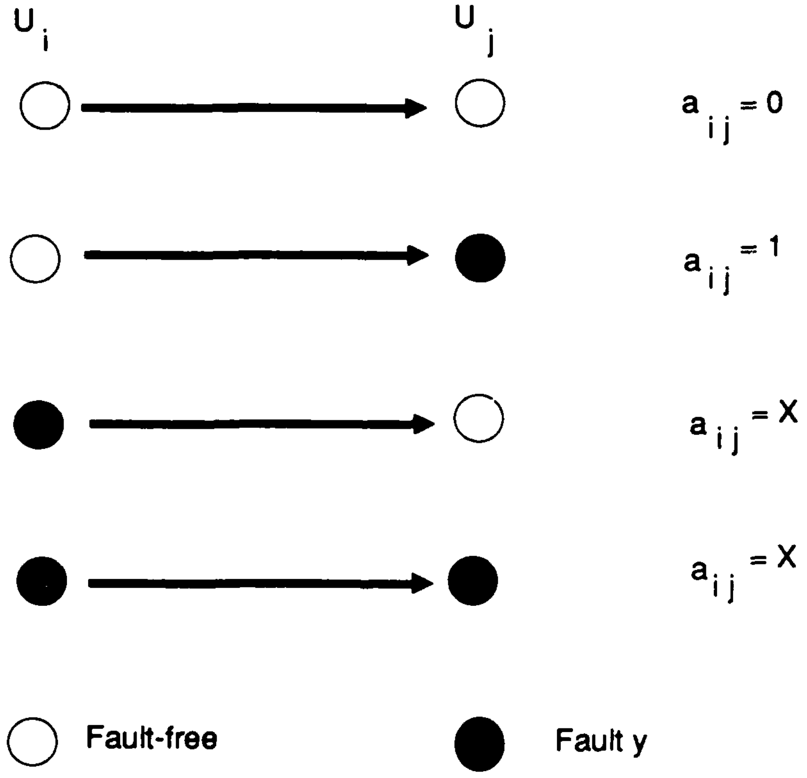
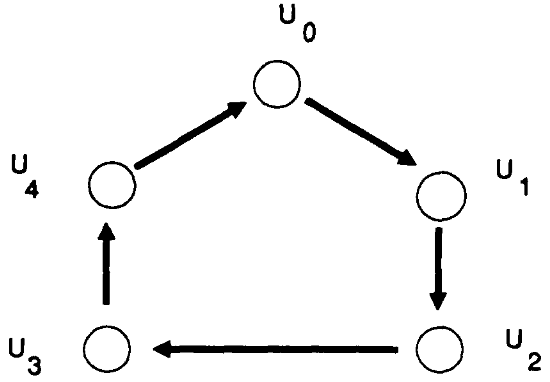


Figure 2.2 Assumed test outcomes in Preparata-Metze-Chien Model



	a_{01}	a_{12}	a_{23}	a_{34}	a_{40}
a)	X	0	0	0	1
b)	1	X	0	0	0
c)	X	X	0	0	1

Figure 2.3 A system and associated test outcomes

Faults in units U_i and U_j are distinguishable if the syndromes associated with them are different. The two faults are indistinguishable if the syndromes associated with two different faults are the same. These definitions may be directly extended to distinguishable and indistinguishable sets of faults called **fault patterns**. Figure 2.3 depicts a system and its test outcomes for three different cases. If U_0 is faulty, the syndrome shown in line a is produced. If U_1 is faulty, the syndrome shown in line b is produced. They are distinguishable since the value a_{40} is different. The multiple fault pattern (U_0, U_1 are faulty) has the syndrome in line c, and since it may be the same as the syndrome for faults $\{U_0\}$ (depending on the unpredictable values of a_{01} and a_{12}), $\{U_0\}$ and $\{U_0, U_1\}$ are indistinguishable.

B. ONE-STEP T-FAULT DIAGNOSABLE SYSTEMS

Definition 2: A system of n units is **one-step t-fault diagnosable** if all faulty units within the system can be uniquely identified, provided the number of faulty units present does not exceed t [Ref. 3].

1. NECESSARY AND SUFFICIENT CONDITIONS:

In this section we investigate the relationship between n and t (the number of faulty units), for one-step diagnosable systems.

Theorem 1: If a system with n units is one-step t-fault diagnosable, then $n \geq 2t+1$. Conversely, if $n \geq 2t+1$, it is always possible to provide a connection to form a system that is one-step t-fault diagnosable [Ref. 3].

Proof: To prove the converse, we construct a maximally connected graph, that is, we make a connection among all possible pairs of these n units in both directions. One characteristic of such a graph is that there exists a loop connecting any subset of n units. It is easily verified that given any loop connecting z units with all test outcomes in the loop exhibiting the value **0**, then the z units in the loop are either all faulty or fault-free. In particular, if $z \geq t+1$, all units in the loop must be fault-free. Otherwise, this would violate the hypothesis on the maximum number of faulty units. The location of a loop of t+1 or more fault-free units will essentially have completed the diagnosis process, and any identified fault-free unit will immediately locate all faulty units through direct links. Since the system can have at most t faulty units, it must contain at least t+1 fault-free units; hence the existence of a loop of t+1 or more fault-free units is guaranteed.

For a system with $n < 2t+1$ units and an arbitrary connection, we show the existence of two distinct allowable fault patterns that may result in exactly the same syndrome. An allowable fault pattern for our specific case is any fault pattern with at most t faulty units. We can consider n as odd and even in two separate cases; but both cases are analogous. Assume $n \leq 2t_0$, with $t_0 \leq t$. Consider the case of an even number of nodes. We partition the system into two parts, P_1 and P_2, each with the same amount of units t_0. Suppose all units in P_1 are faulty and all units in P_2 are fault-free. Then, all links between units within P_2 will have a value **0** and all links pointing from units in P_2 to units in P_1 will have a value **1**. Since the units in P_1 are faulty, many possible configurations of values may occur. One such possible configuration is for all links between units in P_1 to have a value **0** and all links pointing from units in P_1 to units in P_2 to have value **1**. From symmetry, it is seen that when all units in P_1 are fault-free and all units in P_2 are faulty, the same pattern of test results may occur. Hence, it is not always possible for the system to differentiate between the two allowable fault patterns and the system is not one-step t-fault diagnosable [Ref. 3: p-850].

2. OPTIMAL DESIGNS FOR ONE-STEP t-FAULT DIAGNOSABILITY:

For this model it has been shown that the number of units n must be at least 2t+1 for a system to be one-step diagnosable. Now we will try to get the lower bound on the number of units that concurrently test a particular unit.

Theorem 2: In a one step t-fault diagnosable system, a unit is tested by at least t other units [Ref. 3: p-850].

Proof: On the hypothesis that the system is one-step t-fault diagnosable, we may assume that $U_1, U_2, \ldots, U_k$ are all the units in the system which test a certain unit U_0 and $k < t$. Consider the case in which $U_1, U_2, \ldots, U_k$ are all faulty. The outcome of the tests performed by these faulty units may, of course, assume arbitrary values. Hence there is no reliable test being performed on U_0, and the two legitimate fault patterns $(U_1, U_2, \ldots, U_k)$ and $(U_0, U_1, U_2, \ldots, U_k)$ neither of which has more than t faults are not distinguishable. Hence according to Definition 2, the system is not one-step t-fault diagnosable. Since a contradiction has been arrived at, the assertion stated in the theorem is proved.

Definition 3: A one-step t-fault diagnosable system is said to be optimal if n = 2t+1 and each is tested by exactly t units [Ref. 3: p-850].

In general, many optimal designs exists for a system. To describe these families of designs D_t, it is convenient to designate the n units by $U_0, U_1, \ldots, U_{n-1}$, and to perform any computation on the subscripts **modulo** n. We will consider a class of designs in which the testing connection at each unit is identical. In fact, whether there is a testing link from u_i to u_j depends entirely upon the value of **l=j-i** (modulo n). A test exists if and only if $1 \leq l \leq t$. Preparata, Metze and Chien [Ref. 2] showed that a design D_t is an optimal one step t-fault diagnosable system.

C. SEQUENTIALLY DIAGNOSABLE SYSTEMS:

Definition 4: A system of n units is sequentially diagnosable if at least one faulty unit can be identified without replacement, provided the number of faulty units present does not exceed t [Ref. 3: p-849].

It is obvious that every system which is one-step t-fault diagnosable is also sequentially diagnosable. But a system which is sequentially diagnosable may not be one-step t-fault diagnosable. In the previous section, we have seen that nt links are required for a system of n units to be one-step t-fault diagnosable (design D_t). The investigation of sequentially diagnosable systems is motivated by the expectation that fewer test links are required in such systems. Theorem 1 is valid for sequentially diagnosable systems also. Hence for any sequentially t-fault diagnosable systems $n \geq 2t+1$.

Theorem 3: There exists a class of designs with N=n+2t-2 that are sequentially t-fault diagnosable [Ref. 3: p-852].

Proof: Consider the following design. First, connect all units U_0, U_1, ...,U_{n-1} in a loop such that for every i there is a link from U_i to U_{i+1} (all subscripts are taken modulo n). Secondly, select a subset S_1 of 2t-2 units from the set (U_1, U_2, U_3, ...,U_{n-2}) and establish a link from each unit of S_1 to U_0. This is shown in Figure 2.4. Let the number of testing signals from S_1 and U_{n-1} to U_0 having the value 0 (1) be n_0 (n_1). The following cases are possible:

Case 1: **n_1>t**. The assumption (U_0 is not faulty) implies that $n_1 > t$ units are faulty, thus violating the hypothesis on the maximum number of faulty units. Therefore $n_1 > t$ implies U_0 is **faulty**.

Case 2: $\mathbf{n_1<t}$. The assumption (U_0 is faulty) implies that, $n_0 > t-1$ more units are faulty. If $n_1 < t$, $n_1+n_2=2t-2$ and assume $n_1=t-1$. So $n_0=2t-2-n_1$. If we put $n_1=t-1$, then $n_0 = t-1$. For $n_1 = t-2, t-3$ .. and so on, $n_0 > t-1$ but this also violates the hypothesis. Therefore $n_1 < t$ implies U_0 to be **not faulty**.

Case 3: $\mathbf{n_1=t}$. Let's consider the set $S'=S_1 \text{ U } U_{n-1} \text{ U } U_0$ for a total of 2t units. If U_0 is not faulty, the set contains $n_1=t$ faulty units; if U_0 is faulty, the system contains U_0 and $n_0 = t-1$ additional faulty units, for a total of t. In both cases the set contains t faulty units. We conclude that all units of the system not contained with in the set S' are not faulty and at least one fault-free unit can be identified. Therefore, $n_1 = t$ implies the existence and identification of at least one fault-free unit.

To locate at least one faulty unit we proceed as follows. In case 1, U_0 is the faulty unit. In cases 2 and 3 we have located at least one fault-free unit. To locate a faulty unit we simply travel along the loop of testing links in the direction of arrows. We follow the test signals until we see a 1 for the first time, the unit being tested by this link is faulty [Ref. 3: p-852]. So considering all of the three cases above, we have identified at least one faulty unit; which is necessary and sufficient for sequential diagnosis.

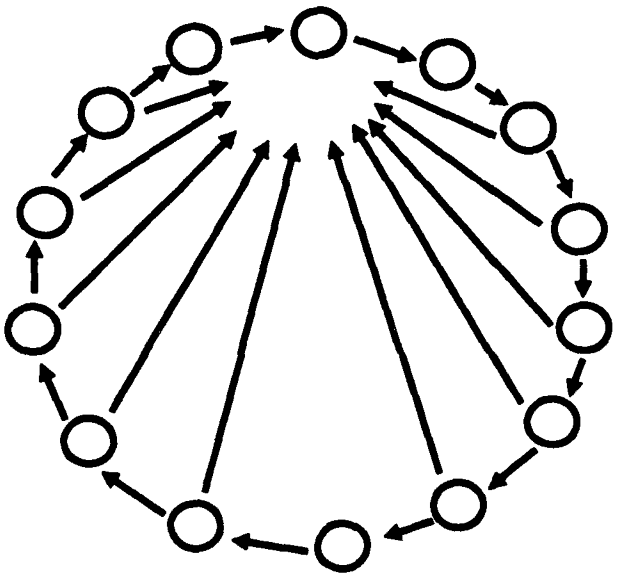


Figure 2.4 An example of sequential diagnosis connection for n=14 and t=6

D. GENERALIZATION OF FAULTS

t_p-fault diagnosability: A system is t_p-diagnosable if and only if the application of the test set identifies precisely which faults are present, provided the number of faults does not exceed t_p [Ref. 9]. (This is precisely one-step t-fault diagnosability.)

The major part of the self-diagnosability of systems has assumed that only permanent (solid) faults can be present. Consideration of intermittent faults is generally difficult since it requires a modeling of the behavior of these faults in a system and also requires interactive testing strategies to detect faults. Mallela and Masson [Ref. 10] consider the effect of intermittent faults in diagnosable systems. The existence of both permanent and intermittent faults in a system, for example, affects the test outcome which is received after repeated applications of the test routines. This outcome may generate an incomplete diagnosis of faulty units, since not all the faulty units in the system may be detected.

t_i-fault diagnosability: A system is t_i-fault diagnosable if in the presence of t_i intermittent faults no fault-free unit will ever be diagnosed as faulty, and diagnosis will be at worst case incomplete [Ref. 4].

In general, the fact that a system is $\mathbf{t_p}$-fault diagnosable does not necessarily imply that it is also $\mathbf{t_i}$-fault diagnosable. Mallela and Mason also give necessary and sufficient conditions for one-step t_i-fault diagnosability.

t/s-diagnosability: A multiprocessing system is t/s-diagnosable if one can always identify a set of processors of size s or less which contains all permanently faulty processors, provided there are no more than t-faulty processors. In general, t < s, and so there is a relaxation of restriction in previous studies that no fault-free processors can be replaced [Ref. 7].

E. SMITH'S ALGORITHM:

Consider three replacement algorithms [Ref. 8] for faulty processors:

ST_1: At each step perform the tests and replace processors which fail at least one test, with randomly chosen spares. If all test results are pass, the system is assumed to be correct.

ST2: At each step, perform the tests and replace processors which fail the maximum number of tests. Replaced processors are placed back into the set of spares. If all test results are pass, the system is assumed to be correct.

ST3: At each step, perform the tests and replace processors which fail the maximum number of tests. Put these into the **SPARE-II** and replace them with randomly selected spares in **SPARE-I**. If the number of processors in **SPARE-I** are not sufficient, then choose any additional needed spares randomly selected from **SPARE-II**. If all test results are pass, the system is assumed to be correct (initially, all spares are in SPARE-I and SPARE-II is empty).

ST1 is fast but tends to replace many fault-free processors (those which fail at least one test by fault-free processors). ST2 replaces fewer fault-free processors, but it is slower. ST3 is the most sophisticated, since it tends to maintain an enrichment in the set of fault-free processors, and resorts to selection of suspected faulty spare processors only when necessary [Ref. 8].

d-disabling rule: Processor U_i is disabled (e.g: not allowed to participate in computation) if and only if U_i fails **d** or more tests by **enabled** processors [Ref. 7]

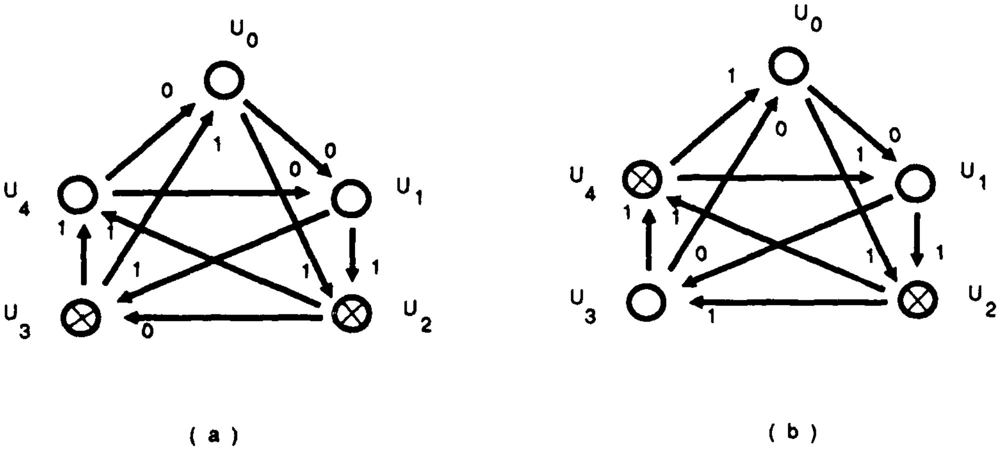


Figure 2.5 Five processor multiprocessor system for two arrangements of faulty processors

Consider the 1-disabling rule in Figure 2.5(a) and assume U_2 and U_3 are faulty and enabled. Then U_4 is disabled even though it is fault-free. U_0 is also fault-free and disabled. However, since U_1 fails no test and it will become enabled permanently. It follows that U_2 and U_3 will eventually be disabled. Thus fault-free nodes U_4 and U_0 which were originally disabled will become enabled permanently. Consider the system in Figure 2.5(b), where there are also two faulty units, and assume the 1-disabling rule applies as before. If U_2 and U_4 are enabled, before any of the processors are enabled, the fail test outcomes they produce disable U_0, U_1 and U_3. Since all fault-free processors are disabled and the tests among faulty processors are pass, both faulty processors are **enabled**. Unlike the case just discussed, the system will never correct itself. Thus, a permanent situation exists where **all faulty processors are enabled and all fault-free processors disabled**. In the same figure, if we apply the **2**-disabling rule with the same initial conditions (e.g: U_2, U_4 are faulty and enabled), the fault-free processors will eventually become disabled, while only one of the faulty processors will be disabled. Thus, the 1-and 2- disabling rule lead to an unsatisfactory diagnosis.

III. PROBLEM

A. SIMPLE DIAGNOSABILITY TESTS FOR MULTIPROCESSING SYSTEMS

Recall that we are interested in distributed fault diagnosis of the system, since ultra reliability can be achieved less expensively. The basic idea behind distributed self-diagnosis is that the diagnosis algorithm is executed on the remaining intact units of the system. In contrast to the central diagnosis which assumes an external (perfect) unit for computing diagnosis results, distributed diagnosis is performed throughout the system. First, a node is diagnosed by its immediate neighboring nodes. In a second step, these local diagnosis results are used to disable processors.

To achieve distributed fault diagnosis in a system, each unit is equipped with disabling circuitry. Thus, testing processors can determine the status of the tested processor. The problem of identifying how many faulty processors can be tolerated before it is impossible to correctly identify them is a very difficult task in general multiprocessing systems. For example, in some cases as is shown in Chapter II, Figure 2.3, the two different fault patterns produce the same test outcome (syndrome).

The problem of locating faulty processors within a multiprocessor system by temporarily halting normal operation and placing it in a **diagnostic mode** has been studied using the **PMC** model. When the number of modules in the system is large, some of them will be idle at a given moment. A test may be any sort of check by one processor on the operation of the other, including applying test vectors and checking resulting outputs. In a concept introduced by Nair, Metze, Abraham [Ref. 9] called **"roving diagnosis"**. One part of the system diagnoses a second part, while the remainder of the system continues normal operation. The part most recently diagnosed as fault-free then takes its turn in diagnosing other parts. Thus, there appears to be a subsystem of diagnosing and diagnosed units which **"roves"** through the system until no parts of it remains undiagnosed. However roving diagnosis, must ensure that first diagnosis will produce unique, identifiable results. The checks are performed at the system level on data elements that constitute the results of computations on these systems. It is assumed [Ref. 10: 298] that each processor has a local memory on which it performs reads and writes.

In addition, it can communicate with other processors in the system through the buffers at various input and output ports. A processor cannot read or write from any other processor's local memory even in the presence of a fault. A fault is any condition that causes a malfunction in a single processor while performing operations.

B. RECONFIGURATION

Definition 5: A system is **c-correctable** using the d-disabling rule if and only if:

1. All faulty nodes are eventually permanently disabled.

2. All fault-free processors are eventually permanently enabled provided there are c or fewer faulty nodes [Ref. 7].

The main goal in system configuration is to **switch-in** all fault-free units and to **switch-out** all faulty units. But this switching is not between two working systems, just between working system and spares. The goal is not only to **switch-out** the faulty units but also keep the working system functional. That gives more flexibility to the system but increases the cost. The problem is to derive a distributed strategy for correct switching which is insensitive to the arrangement of faulty processors. Sometimes it may be difficult to replace a specific processor, so rearrangement of applied tests can give more accurate results. A flexible test arrangement will allow an approach which views the diagnostic task as one of arranging processors into two groups, a working group and a spare group. Another approach is to have three groups, one group for critical operations, one for noncritical operations, and one for spares. However in this thesis, we will consider only the first approach.

C. RELATIONSHIP BETWEEN ENABLED/DISABLED UNITS AND SYSTEM RELIABILITY

In an implementation of distributed diagnosis, to have correct diagnostics, two major important problems must be considered:

1. Reliable implementation of the disabling criteria and function.

2. Reliable transmission of appropriate test (pass, fail) and result signals of disabling criteria (enabled or disabled) for system units.

It should be noted that in distributed diagnosis, only local information is used to identify faulty processors. In central diagnosis all test results are used. Thus, we would expect distributed diagnosis to be less accurate. This manifests itself in a fewer number of faulty nodes which can be tolerated in distributed diagnosis.

IV. METHOD OF APPROACH

A. WHY A CAD-TOOL?

Our approach to the problem of developing diagnosis strategies is to develop a **CAD** (Computer Aided Design) tool for the simulation of different fault patterns and different reconfiguration strategies. Previously all studies have used hand calculations for this purpose. When the number of units in the system has increased to more than seven, hand calculations becomes complex. Thus, the user can only simulate a limited number of units and fault patterns. Using the CAD-tool, the user can simulate from 2 to 20 units with various fault patterns. The restriction of 20 units is due to limitations of the monitor screen.

Thus, the tool facility gives the user an opportunity of simulating a large number of units and fault patterns in a system. The number of units in a network is known in advance and can be predefined in to the tool-program. The names and number of faulty nodes are determined by the user. Testing connections can be predefined by the user or the program. Only the test procedure (worst_case or user_defined_case) can be chosen by the user. Also the user defines the disabling criteria. After input by the user, the **CAD-tool** determines test results, disabled, enabled units and then displays the system in a control unit monitor. By using the CAD-tool, a computer network is automatically controlled without any hand calculation.

B. TOOL DEFINITIONS:

This CAD-tool is written in the **C** programming language [Ref. 12] using **PMC** graph model. The terms used in the program are listed below and given short explanations:

N=The number of units in the system (may change from 1 to 20).

f=The number of faulty nodes($0 \leq f \leq N-1$).

T=The number of units which tests one unit. This number is the same for all units. Test results according to test connection are determined by the program reflecting the user desire as a worst-case or arbitrary case.

For the worst-case, the program itself determines all test results. That is, faulty testing units produce fail (1) test outcome for fault-free and pass (0) test outcome for faulty tested units. This information is completely opposite to the status of the units. This is the reason it is called worst case. For the user defined (arbitrary) case, test outcomes for **faulty testing units** (for faulty or fault-free tested units), are defined by the user.

d=Is the disabling criteria which is defined by the user. If a tested unit has, at least **d** fail test outcomes by **enabled** units, the unit will be disabled.

C. TOOL SPECIFICATION

Figure 4.1 shows the flowchart of the main body of the system tool. As can be seen, the user can specify initial conditions and then allow the system to execute diagnostic steps one after the other.

Figure 4.2 shows a more detailed flowchart of the program. First, the user defines the number of units in the system. If this number is less than 0 or greater than 20, the program produces an error message. The user defines the number and the names of faulty nodes. Next, the user defines **T** (the number of units testing one unit) and the test procedure (as worst case or arbitrary case). The program determines the test results and displays them onto the screen. The user defines the disabling criteria, the number and names of enabled units (all units are disabled initially). The tool displays the whole system in the initial conditions by calling the subroutine drawing.

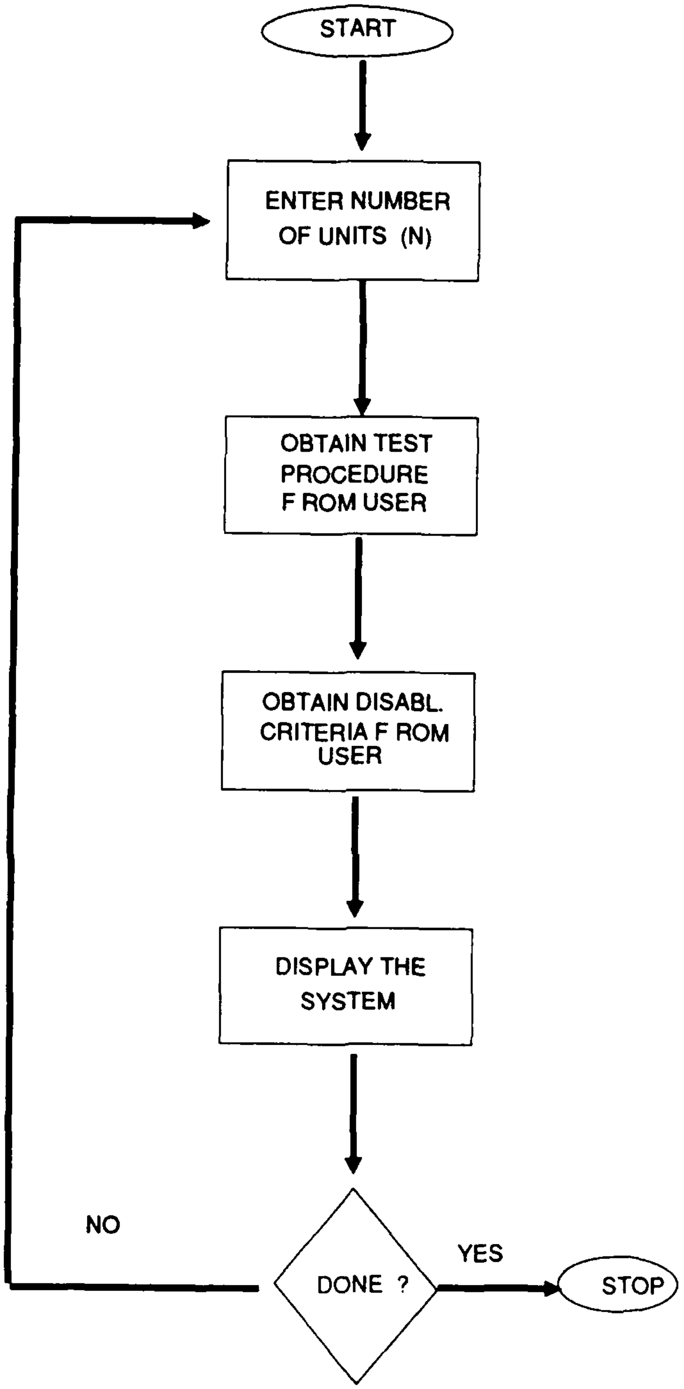

START
ENTER NUMBER OF UNITS (N)
OBTAIN TEST PROCEDURE F ROM USER
OBTAIN DISABL. CRITERIA F ROM USER
DISPLAY THE SYSTEM
NO
DONE ?
YES
STOP

Figure 4.1 Flow chart of CAD-tool

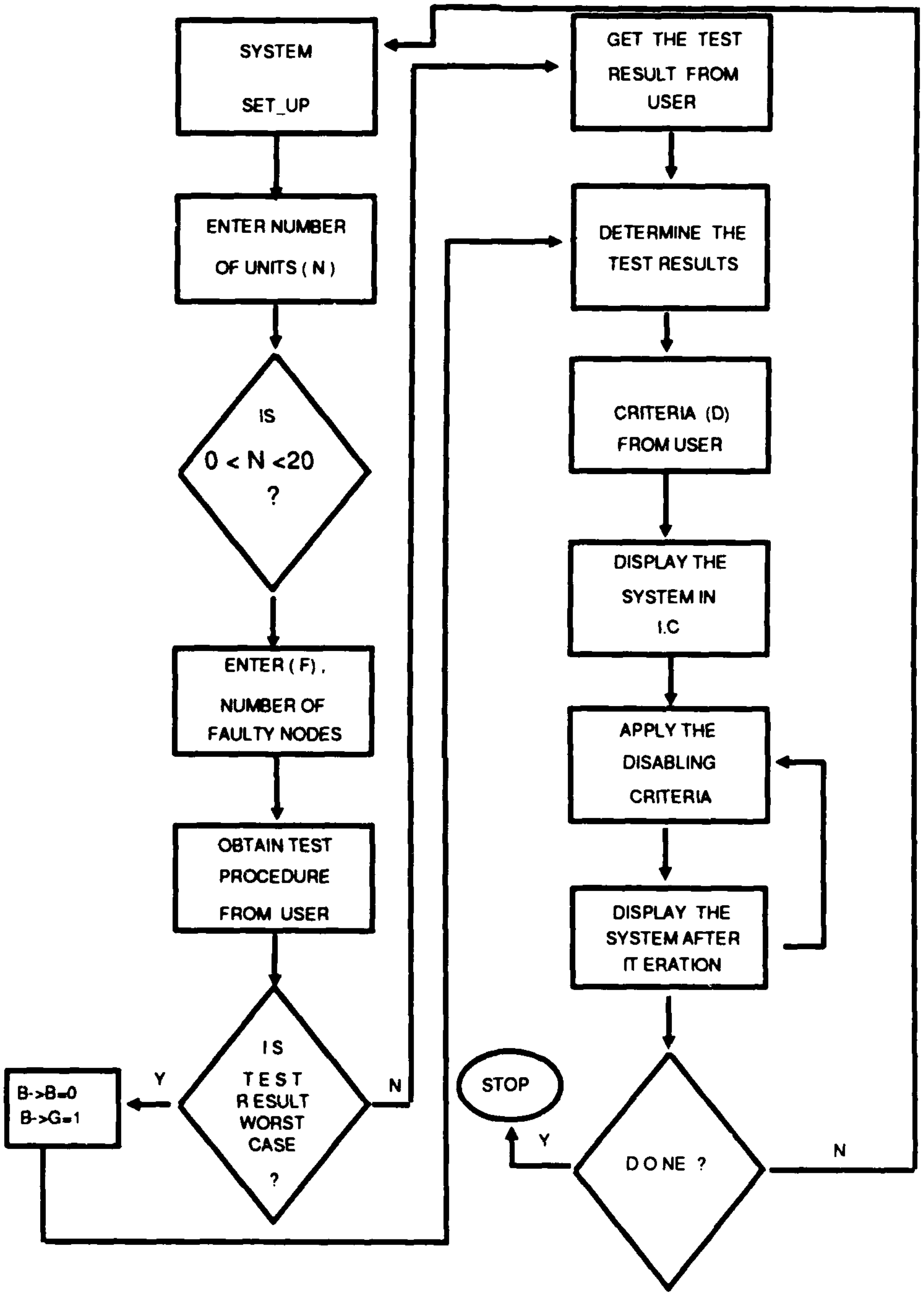

SYSTEM
SET_UP
ENTER NUMBER OF UNITS (N)
IS 0 < N <20 ?
ENTER (F), NUMBER OF FAULTY NODES
OBTAIN TEST PROCEDURE FROM USER
IS TEST R ESULT WORST CASE ?
Y
N
B->B=0
B->G=1
GET THE TEST RESULT FROM USER
DETERMINE THE TEST RESULTS
CRITERIA (D) FROM USER
DISPLAY THE SYSTEM IN I.C
APPLY THE DISABLING CRITERIA
DISPLAY THE SYSTEM AFTER IT ERATION
DONE ?
Y
N
STOP

Figure 4.2 Detailed flow chart of CAD-tool

To see the application of the disabling rule, the user selects option #5 from the menu shown in Table 4.1. Then, the program determines the **enabled** and **disabled** units and displays the first iteration by calling the **drawing** subroutine. The user can go onto more iterations with the same conditions. After some number of iterations, the user can exit the program or go back to the beginning, where he/she can simulate another system with another conditions.

1. INTRODUCTION
2. SYSTEM SET_UP
3. SET TEST RESULTS
4. SET THE DISABLING CRITERIA
5. APPLY DISABLING RULE
6. EXIT

Table 4.1 Menu of CAD-tool

D. TOOL REALIZATION

The CAD tool is made up of five main parts (subroutines). The first, menu option #1, gives a brief explanation of the program. Option #2 sets up the type of system, number and names of units, number and names of faulty units. Option #3 sets up **T**, and test procedure. Option #4 sets up the disabling criteria, number and the names of the enabled units. Then it displays the system initial conditions calling the subroutine drawing. Option #5 applies the disabling criteria and determines the enabled and disabled units, then it displays the system. In the drawing subroutine, enabled fault-free units are green, enabled faulty nodes are also green with X's inside circles. Disabled fault-free nodes are red and disabled faulty nodes are red with X's inside circles. Test results are represented by the color of testing arrows. A green arrow means a **pass (0)** test outcome, and a red arrow means **fail (1)**. Each time, after going through each option, the menu comes onto the screen. So if the user makes a mistake somewhere in the program, he/she can correct it easily, choosing the same option from the menu. The main part of program is very straightforward and just calls the subroutines according to selected menu options.

V. RESULTS

Figure 5.1 shows a photograph of the CAD-tool menu. Figures 5.2 through 5.5 shows the initial condition and three step iterations of a five unit multiprocessor system. In this system U_2 and U_3 are faulty and enabled initially and shown with color **green**; other units are disabled and shown with color **red**. The disabling criteria is 1 and the test results are the worst case. After the first iteration units U_0 and U_4 are disabled (red) and all the other units are enabled (green). After the second iteration U_1 is enabled and all the other units are disabled. After the third iteration, all faulty units are disabled (U_2, U_3) and all fault-free units are :nabled. In this case, the 1-disabling criteria gives the desired results. This example is explained in Appendix B as Case 1.

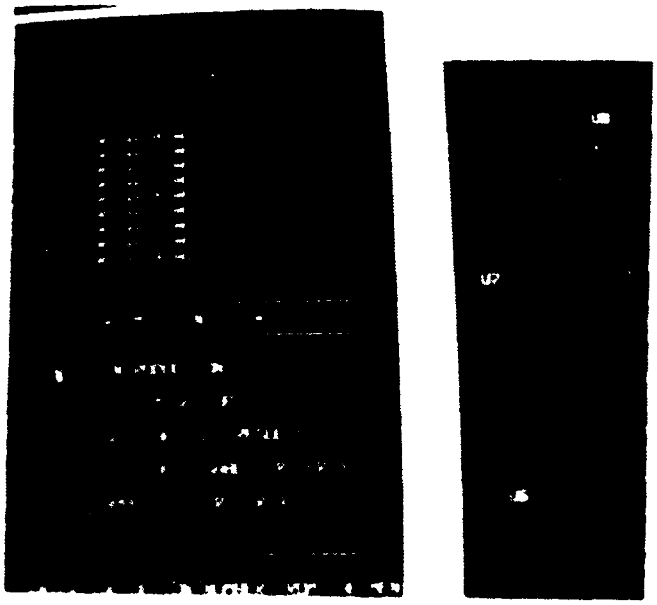


Figure 5.1 CAD-tool menu and test outcomes

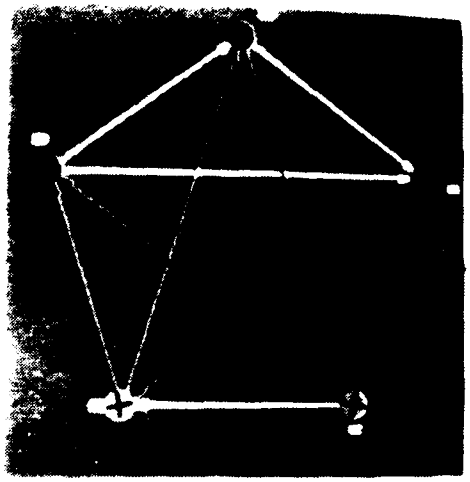


Figure 5.2 Initial condition

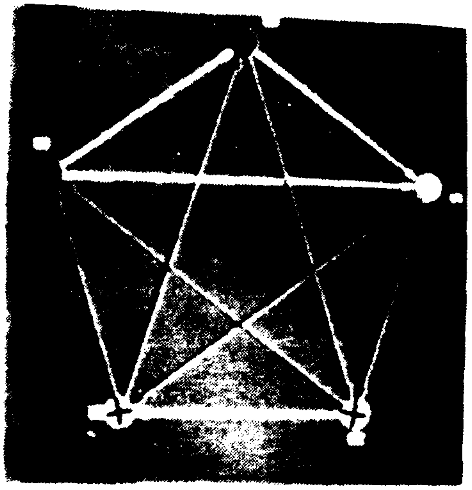


Figure 5.3 First iteration

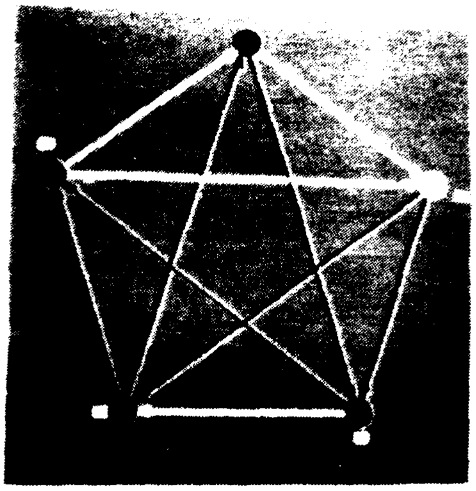


Figure 5.4 Second iteration

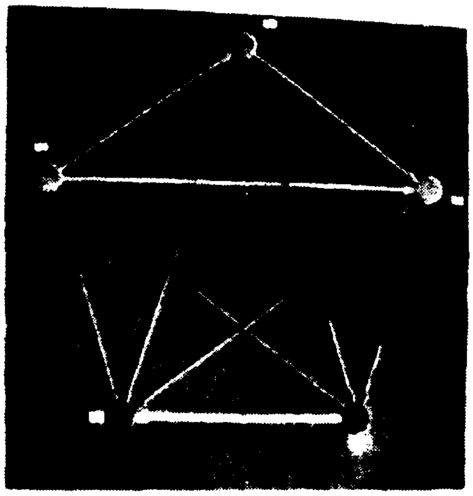


Figure 5.5 Third iteration

Figures 5.6 through 5.9 show another five unit multiprocessing system. In this example, U_1 and U_4 are faulty and enabled initally. Disabling criteria is 2 and test results are also **worst case**. After the first iteration all units are enabled. After the second iteration only U_4 is disabled and all the other units are enabled. Figure 5.8 and Figure 5.9 both are the same. This means that the system stays in that state and cannot correct itself. This example is explained in Appendix B as Case 3.

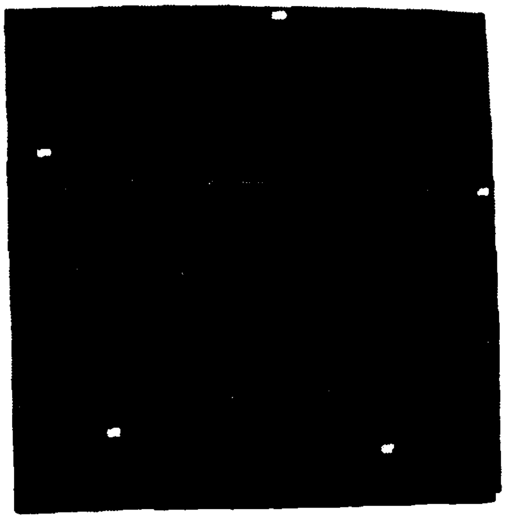


Figure 5.6 Initial condition

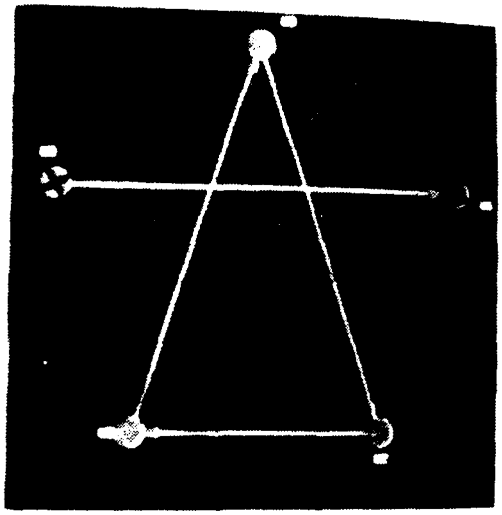


Figure 5.7 First iteration

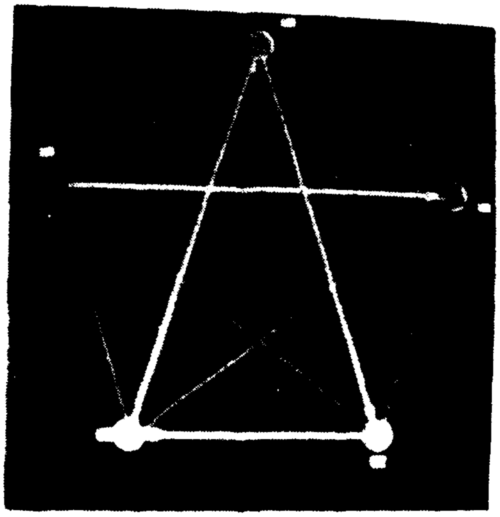


Figure 5.8 Second iteration

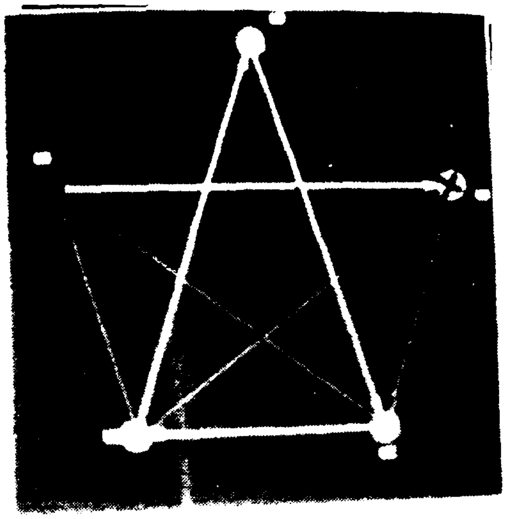


Figure 5.9 Third iteration

Figures 5.10 through 5.14 show a seven unit multiprocessor system. Ii. this system, U_1, U_3, U_5 are faulty units and enabled initially. Test results are also worst case and disabling criteria is 2. After the first iteration, U_4 and U_6 are disabled, all the other units are enabled. After the second iteration U_3, U_4, U_6 are disabled and the other units are enabled. After the third iteration only U_3 is disabled. After the fourth iteration all faulty units are disabled and all fault-free units are enabled. This indicates the 2-disabling criteria works and the system corrects itself. This example is explained in Appendix B as Case 6.

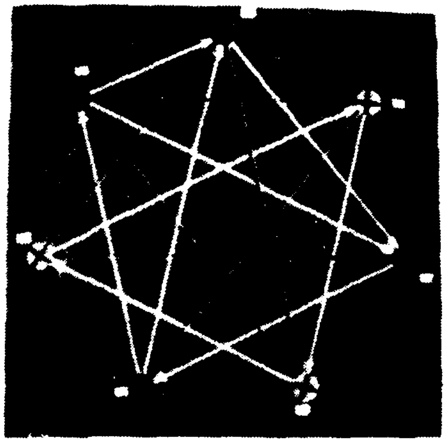


Figure 5.10 Initial condition

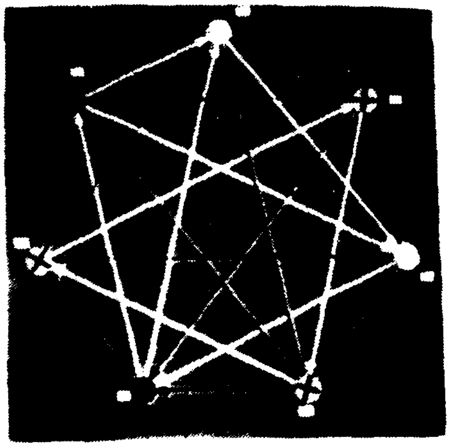


Figure 5.11 First iteration

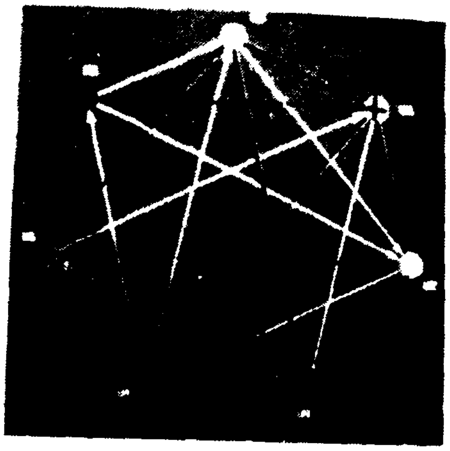


Figure 5.12 Second iteration

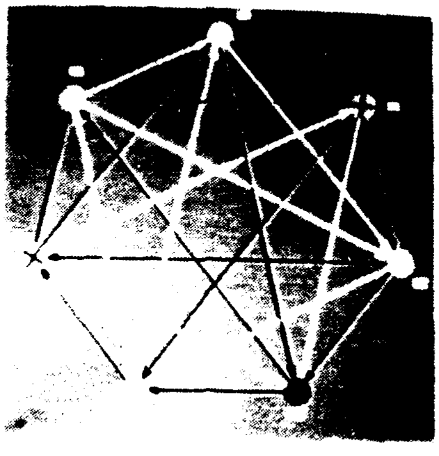


Figure 5.13 Third iteartion

Figure 5.14 Fourth iteration

Figures 5.15 through 5.20 show a six unit system. In this system U_1, U_3, U_5 are faulty units and the disabling criteria is 2. Test results are arbitrary (user defined) and are defined as followes: faulty testing units produce fail (1) test outcome for faulty tested units and produce pass (0) outcome for fault-free tested units. In this example, faulty units are alternately disabled and enabled. Thus the system will never correct itself. It displays an oscillation of period six. This example is explained in Appendix B as Case 19.

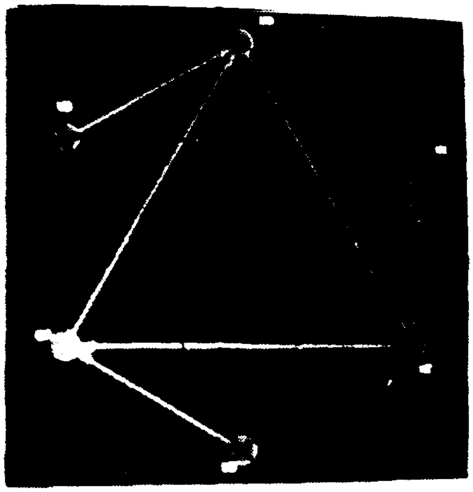


Figure 5.15 Initial condition

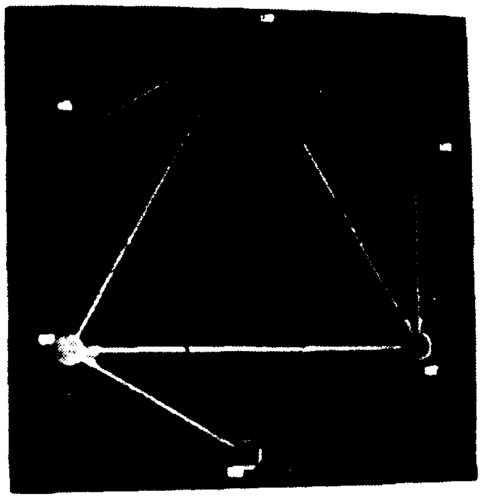


Figure 5.16 First iteration

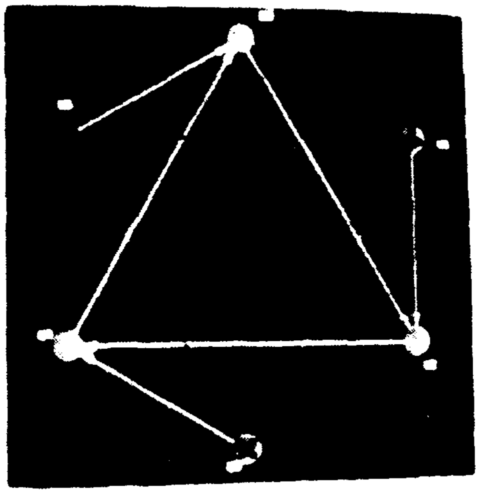


Figure 5.17 Second iteration

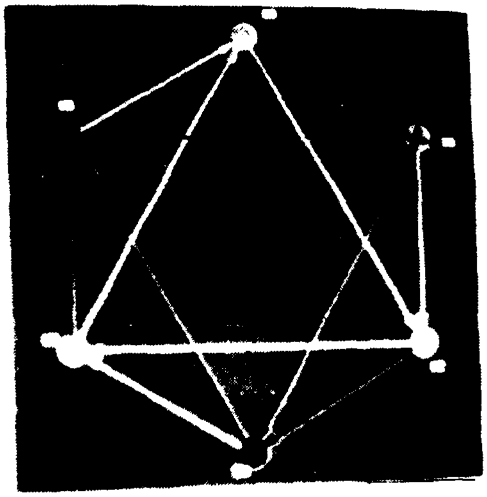


Figure 5.18 Third iteration

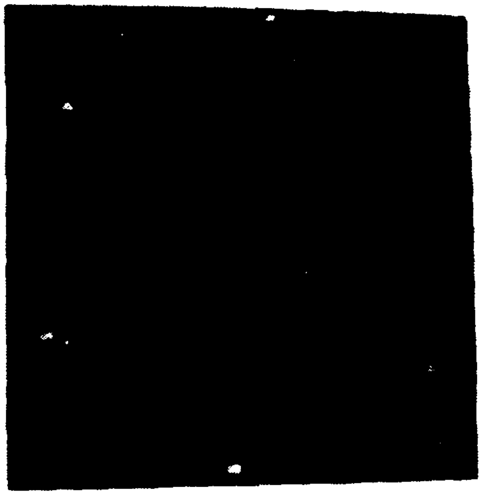


Figure 5.19 Fourth iteration

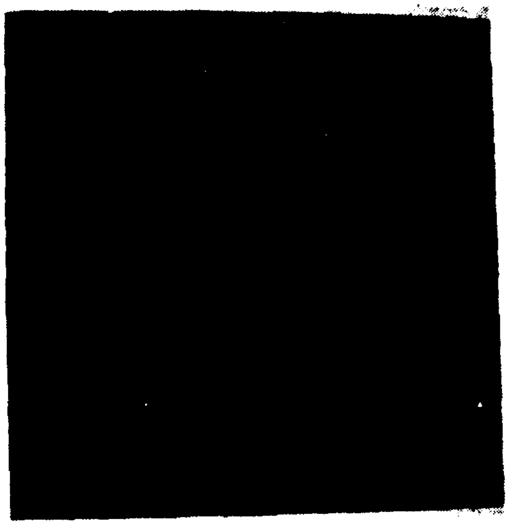


Figure 5.20 Fifth iteration

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSION

This thesis introduces distributed diagnosis. The analysis of distributed diagnosis is difficult without a CAD tool. In this research, a CAD-tool has been developed based upon the PMC graph model. Using this tool, the user can simulate various number of configurations and fault patterns. The tool provides a step by step procedure for user to follow. In this tool, the information related to the faulty nodes (the numbers and the names of faulty nodes) is provided by the user. Then the user simulates the system as much as wanted.

In the CAD-tool, fail test outcomes by enabled porcessors for each unit are counted and compared with the disabling criteria. If fail test outcomes exceed the criteria, then the unit is disabled. Unlike the central diagnosis algorithm which eventually settled on a final arrangement of processors, the algorithm denoted here develops dynamic behavior.

B. RECOMMENDATIONS

It is expected that this tool will be used to study optimum disabling criteria for various systems. For example, we hope that it will free the user of the tedium of generating examples, allowing him to prove properties of the system. One possibility is that it could be used in a knowledge base system, which would be used to prove properties of the disabling criteria.

APPENDIX A

SOURCE CODE

```
/* This menu helps the user to determine the main selections of the program.
If the user wants to run the program for very  FIRST TIME  should choose the
option #2.To choose INTRODUCTION is outside this restriction.*/

 char fault_array[20],disable_array[20],dis_res_array[2]
 char U;
 int  test_array[20][20];
 int  N,fmax,f,T,k,j,i,no_units_set,p,w,l,dis_crit,count;
 int  no_en_set;
 int  response;
      menu()
        int response;
         {
          printf("  -------------------------------- \n");
          printf(" |        = M E N U =            |\n");
          printf(" |-------------------------------|\n");
          printf(" |                               |\n");
          printf(" |    1. INTRODUCTION            |\n");
          printf(" |                               |\n");
          printf(" |    2. SYSTEM SET_UP           |\n");
          printf(" |                               |\n");
          printf(" |    3. SET THE TEST RESULTS    |\n");
          printf(" |                               |\n");
          printf(" |    4. SET THE DISABL. CRITERIA |\n");
          printf(" |                               |\n");
          printf(" |    5. APPLY DIS.CRITERIA      |\n");
          printf(" |                               |\n");
          printf(" |    6. EXIT                    |\n");
          printf(" ------------------------------- \n\n");

          printf("ENTER THE OPTION NUMBER FROM THE MENU \n\n");
            }
introduction()

 {
printf("************************************************\n");
printf("*                                              *\n");
printf("*      THESIS TOPIC: FAULT TOLERANT COMPUTING  *\n");
printf("*                                              *\n");
printf("*      IN DISRIBUTED COMPUTER NETWORKS.        *\n");
printf("*                                              *\n");
printf("*     Author: Ibrahim  DINCER                  *\n");
printf("*                                              *\n");
printf("*     Thesis Advisor:  Prof. Jon  T.  BUTLER   *\n");
printf("*                                              *\n");
printf("*       NAVAL    POSTGRADUTE    SCHOOL         *\n");
printf("*                                              *\n");
printf("*       ELECTRICAL AND COMPUTER ENGINEERING    *\n");
printf("*                                               *\n");
printf("*        EXTENSION :3299                        *\n");
printf("*                                               *\n");
printf("*          DATE  : APRIL 23,1987                *\n");
printf("************************************************\n");
printf(" This program is for simulation of distributed   \n");

printf("diagnosis algorithm in a computer network.For this\n");

printf(" purpose PREPARATA_METZE_CHIEN is used. The number\n");
printf("                                                  \n");
printf(" of nodes in the system is restricted TO NO MORE  \n");
printf("                                                  \n");
printf("THAN 20.The user enters the number of nodes,faulty\n");
printf("                                                  \n");
printf("nodes in the network,test procedure and disabling \n");
printf("                                                  \n");
printf("criteria.The program displays the network,test    \n");
printf("                                                  \n");
printf("outcomes and shows enabled fault_free nodes and   \n");
printf("    disabled faulty nodes.                        \n");
printf("   N= NUMBER OF NODES IN THE SYSTEM               \n");
printf("                                                  \n");
printf("   D=DISABLING CRITERIA FOR FAULTY NODES          \n");
printf("                                                  \n");
printf("   F= NUMBER OF FAULTY NODES IN THE SYSTEM        \n");
printf("                                                  \n");
printf("FMAX=NUMBER OF ALLOWED FAULTY NODES IN THE SYSTEM \n");
printf("                                                  \n");
printf("T= NUMBER OF UNITS WHICH ARE TESTING ONE UNIT     \n");

   }

 /* THIS  SUBROUTINE DEFINES  THE NAMES OF NODES AND ALSO DEFINES THE FAULTY
NODES  IN THE SYSTEM  */
   units()
      {
       printf(" THE UNITS OF THE SYSTEM ARE\n\n");
       for(i=0; i<N; ++i)
         {
          printf("%c%d,",'U',i);
           }
        printf("\n");
        printf(" ENTER THE NUMBER OF FAULTY NODES \n");
        scanf("%d",&f);
 /* THIS TWO LOOPS KEEP THE USER IN THE ALLOWED  LIMITS FOR
  FAULTY UNITS*/
        while(f<=0 ¦¦ f>N)
           {
            printf(" 'F' SHOULD BE  GREATER THAN  ZERO ");
            printf("  AND LESS THAN N \n");
            scanf("%d",&f);
           }
       printf("THERE ARE  %d FAULTY NODES  \n\n",f);
       j=1;
/* INDICATES THE ARRAY TO DEFINE THE FAULTY NODES */
       for(i=0; i<=N-1; ++i)
        {
         fault_array[i]='G';/*INITIALLY ALL NODES ARE GOOD*/
         }
       no_units_set=1;
       printf("ENTER THE FAULTY UNIT NUMBER ONE AT A TIME \n");
       while(no_units_set <=f)/*REPEAT UNTIL 'F' UNITS ENTERED
         {
          scanf("%d",&i);
          while(i>(N-1) || i<0)
            {
             printf(" UNIT  NUMBER IS NOT VALID, TRY AGAIN ]\n");
             scanf("%d",&i);
            }
          if (fault_array[i!=='B')
            {
             printf("THIS UNIT IS PREVIOUSLY DEFINED AS ");
             printf("  FAULTY,TRY AGAIN ]\n\n");
            }
          else
            {
             fault_array[i] = 'B';
             printf(" FAULTY UNIT # %d IS  U%d \n\n",j,i);
             ++j;
             ++no_units_set;
             }
              }
           }



/* THIS SUBROUTINE SETS UP THE SYSTEM TO BE TESTED */


 sys_set_up()
     {
       printf(" TO DETERMINE THE NETWORK ENTER ONE OF THE ");
printf(" OPTIONS BELOW\n");
       printf("\n");
       printf("       1.DESIGN  \n\n");
       printf("       2.ARBITRARY SYSTEM \n\n");
       scanf("%d",&p);
       printf("p=%d\n\n",p);

       if  (p==1)
        {
         printf("ENTER THE NUMBER OF NODES IN THE SYSTEM\n\n");
         scanf("%d",&N);
```

```
          while(N>20 ¦¦ N<=0)
           {
            printf("THE NUMBER OF UNITS IS NOT VALID,");
            printf(" TRY AGAIN \n");
            scanf("%d",&N);
            printf("N=%d\n\n",N);
            }
           units();
           }
           else
            {
             printf(" THIS SYSTEM WILL BE DEFINED   LATER \n\n");

             }
         }


 /*  THIS  SUBROUTINE  DETERMINES  THE  TEST  RESULTS  FOR THE SYSTEM.IN THE
'WORST_CASE',PROGRAM DETERMINES ALL THE TEST RESULTS; FOR THE ARBITRARY CASE
TEST RESULTS  FOR THE TESTED UNITS BY 'FAULTY' TESTING UNITS WILL BE DEFINED
BY THE USER. */

 test ()
  {
   printf(" 'T' IS THE NUMBER OF UNITS TESTING ONE NODE;ENTER");
   printf("'T'\n");
   scanf("%d",&T);
   printf(" DO YOU WANT 'WORST_CASE' TEST  RESULTS?IF  YES,ENTER");
printf("1\n");
   scanf("%d",&w);
   printf("w=%d\n",w);
   if (w==1)
     {
      for (j=1;j<=T;++j)
       {
        for (k=0;k<=N-1;++k)
         {
          l=k-j;
          if (l<0)
            {
             l=l+N;
             }
          if((fault_array[k!=='B') &&(fault_array[l!=='B'))
            {
             test_array[k![j!=0;
             }
          else if((fault_array[k!=='B') ¦¦(fault_array[l!=='B'))
           {
            test_array[k![j'=1;
            }
          else
           {
```

```
           {
            test_array[k![j!=0;
            }
          }
        }
      }
   else /*THIS PART user_defined ARBITRARY TEST RESULTS */
    {
     for (j=1;j<=T;++j)
      {
       for (k=0;k<=N-1;++k)
        {
         l=k-j;
         if (l<0)
           {
            l=l+N;
            }
         if (fault_array[l]=='B')
           {
            printf("TEST RESULT NODE #%d BY NODE # %d IS  ",k,l);
            scanf("%d",&test_array[k][j]);
            while(test_array[k][j]=0 && test_array[k][j]=1)
             {
              printf("TEST RESULTS SHOULD BE 0 OR 1 \n");
              scanf("%d",&test_array[k][j]);
              }

printf("test_array[%d][%d]=%d\n",k,j,test_array[k][j]);
            }
            else if ( fault_array[k]=='B')
             {
              test_array[k][j]=1;
               }

            else
             {
              test_array[k][j]=0;
               }
             }
           }
         }
    for(k=0;k<=N-1;++k) /*THIS PART PRODUCES TEST_RESULT MATRIX */
     {
      for(j=1;j<=T;++j)
       {
        printf("  %d  ",test_array[k][j]);
         }
        printf("\n\n");
       }
   }  /*  END OF TEST SUBROUTINE */

 /*"THIS PART OF PROGRAM IS DRAWING THE NETWORK FOR DISPLAY" */
```

```
#include <device.h>
#include <gl.h>
#define resetls TRUE


drawing()
 {

  int i,j,k,x,y,x1,y1,x2,y2,x3,y3,x4,y4,x5,y5;
  int x6,y6,x7,y7,x8,y8,t,r,R;
  char number[20],Z;
  float pi,theta,phi,rho,psi,tau;
  short ang;
  pi=3.1416295;
  ginit();
  viewport(400,1000,100,700);
  cursoff();
  color(BLUE);
  clear();
  linewidth(4);
  ortho2(-350.0,350.0,-350.0,350.0);
  R=300;
  r=20;
  x=R*cos(pi/2);
  y=R*sin(pi/2);
  x3=x+r*cos(5*pi/4);
  y3=y+r*sin(5*pi/4);
  x4=x+r*cos(pi/4);
  y4=y+r*sin(pi/4);
  x5=x+r*cos(7*pi/4);
  y5=y+r*sin(7*pi/4);
  x6=x+r*cos(3*pi/4);
  y6=y+r*sin(3*pi/4);
  for(k=0;k<=N-1;++k)
      {
       i=k+1;
       if(i>=N)
        {
         i=i-N;
         }
       ang=(-3600.0/N);
       rotate(ang,'Z');
       while(getbutton(MOUSE3) ]=1);
       if (fault_array[i]=='B' && disable_array[i]=='D')
          {
           color(RED);
           circfi(x,y,r);
           color(BLACK);
           move2i(x3,y3);
           draw2i(x4,y4);
           move2i(x5,y5);
           draw2i(x6,y6);
```

```
    }
if(fault_array[i]=='B' && disable_array[i]=='E')
   {
    color(GREEN);
    circfi(x,y,r);
    color(BLACK);
    move2i(x3,y3);
    draw2i(x4,y4);
    move2i(x5,y5);
    draw2i(x6,y6);
    }
if(fault_array[i]=='G' && disable_array[i]=='E')
   {
    color(GREEN);
    circfi(x,y,r);
    }
if(fault_array[i]=='G' && disable_array[i]=='D')
   {
    color(RED);
    circfi(x,y,r);
    }
color(WHITE);
cmov2i(x+30,y+30);
sprintf(number,"U%d",i);
charstr(number);
for(j=1;j<=T;++j)
    {
     l=j+i;
     if(l>=N)
      {
       l=l-N;
       }
     if (test_array[l][j]==1)
      {
       color(RED);
       }
      if(test_array[l][j]==0)
       {
        color(GREEN);
        }
      theta=2*pi+(pi/2)-(2*pi/N)*j;
      phi=pi/2-(pi/N)*j;
      rho=pi/2-(2*pi/N)*j;
      psi=(pi/N)*j;
      tau=pi/6;
      x1=r*sin(phi);
      y1=R-r*cos(phi);
      x2=R*cos(theta)-r*cos(phi-rho);
      y2=R*sin(theta)+r*sin(phi-rho);
      x7=x2-r*sin(pi/2-psi-tau/2);
      y7=y2+r*cos(pi/2-psi-tau/2);
      x8=x2-r*cos(psi-tau/2);
```

```
            x7=x2-r*sin(pi/2-psi-tau/2);
            y7=y2+r*cos(pi/2-psi-tau/2);
            x8=x2-r*cos(psi-tau/2);
            y8=y2+r*sin(psi-tau/2);
            move2i(x1,y1);
            draw2i(x2,y2);
            draw2i(x7,y7);
            move2i(x2,y2);
            draw2i(x8,y8);
             }
        while(getbutton(MOUSE1) ]=1);
         }
    gexit();

}
  /* THIS PART DETERMINES  THE INITIAL CONDITIONS AND DISPLAYS

                  THE SYSTEM IN INITIAL CONDITIONS */

 disable()
   {
    printf("ENTER THE NUMBER OF ENABLED  NODES\n");
    scanf("%d",&no_en_set);
    printf("ENTER THE MINIMUM NUMBER OF FAIL TEST RESULTS BY");
    printf("ENABLED PROCESSORS WHICH  DISABLE THE TESTED ");
    printf("PROCESSOR \n");
    scanf("%d",&dis_crit);
    for(i=0;i<=N-1;++i)
     {
      disable_array[i]='D';
       }
     count=0;
     j=1;
     printf("ENTER THE ENABLED  UNIT NUMBER ONE AT A TIME \n");
     while(count<no_en_set) /* repeat until all units are
                                  entered */
       {
        scanf("%d",&i);
        if (i>N-1 ¦¦i<0)
          {
           printf("UNIT NUMBER IS NOT VALID,TRY AGAIN ]\n");
           }
        else if (disable_array[i]=='E')
         {
          printf("THIS UNIT IS PREVIOUSLY DEFINED AS ENABLED,");
          printf("TRY AGAIN ]\n\n");
          }
        else
         {
          disable_array[i]='E';
          printf("ENABLED UNIT #%d IS U%d\n\n",j,i);
          printf("disable_array[%d]=%c\n",i,disable_array[i]);
         }
      drawing();
     }

/* THIS PART OF THE PROGRAM DETERMINES ENABLED AND DISABLED

        NODES AFTER THE ITERATION,DISPLAYS THE SYSTEM */

  apply()
    {
     for (k=0;k<=N-1;++k)
       {
        count=0;
        for(j=1;j<=T;++j)
         {
          l=k-j;
          if(l<0)
           {
            l=l+N;
             }
          if((test_array[k][j]==1) && (disable_array[l]=='E'))
            {
             ++count;
              }
          if (count>=dis_crit)
             {
              dis_res_array[k]='D';
              printf("\n");
               }
           }
        if (count<dis_crit)
          {
           dis_res_array[k]='E';
            }
       }
       for(k=0;k<=N-1;++k)
        {
         printf("dis_res_array[%d]=%c\n",k,dis_res_array[k]);
         }
      for(k=0;k<=N-1;++k)
       {
        disable_array[k]=dis_res_array[k];
         }
      drawing();
      printf("\n\n");
     }
#include "gl.h"
#include <stdio.h>
#include <device.h>

               main()
```

```
 {
   ginit();
   cursoff();
   color(WHITE);
   clear();
   textport(0,350,10,900);
   linewidth(6);
   while(response ]=6)
      {
       menu();
       scanf("%d",&response);

       if (response==1)

         introduction();

       if (response==2)

         sys_set_up();

       if (response==3)

         test();

       if (response==4)

          disable();

       if (response==5)

           apply();
      }
   printf("   PROGRAM  IS OVER  \n");
}
```

APPENDIX B

HAND CALCULATION OF DIFFERENT CASES

Case 1. A five unit multiprocessor system, U_2 and U_3 are faulty units and shown underlined. Test results are **worst case**, disabling criteria is 1. In the matrix shown below testing units are placed on the x axis, tested units are placed on y axis.

$$\begin{array}{ccc}
 & U_4 & \underline{U_3} \\
U_0 & 0 & 1 \\
 & U_0 & U_4 \\
U_1 & 0 & 0 \\
 & U_1 & U_0 \\
\underline{U_2} & 1 & 1 \\
 & \underline{U_2} & U_1 \\
\underline{U_3} & 0 & 1 \\
 & \underline{U_3} & \underline{U_2} \\
U_4 & 1 & 1
\end{array}$$

a. first iteration with I.C

U_1, U_2, U_3 are **enabled**

U_0, U_4 are **disabled**

b. second iteration

U_1 is **enabled**

U_0, U_2, U_3, U_4 are **disabled**

c. third iteration

U_0, U_1, U_4 are **enabled**

U_2, U_3 are **disabled**

* all faulty nodes are **disabled**, all fault-free nodes are **enabled**

Case 2. A five unit multiprocessor system, with U_2 and U_3 are faulty units and enabled initially. Test results are arbitrary (user defined) case and disabling criteria is 1. Arbitrary test results have shown underlined.

	U_4	U_3
U_0	0	$\underline{0}$
	U_0	U_4
U_1	0	0
	U_1	U_0
$\underline{U_2}$	1	1
	$\underline{U_2}$	U_1
$\underline{U_3}$	$\underline{0}$	1
	$\underline{U_3}$	$\underline{U_2}$
U_4	$\underline{0}$	$\underline{1}$

a. first iteration with I.C

U_0, U_1, U_2, U_3 are **enabled**

U_4 is **disabled**

b. second iteration

U_0, U_1 are **enabled**

U_2, U_3, U_4 are **disabled**

c. third iteration

U_0, U_1, U_4 are **enabled**

U_2, U_3 are **disabled**

* all faulty nodes are **disabled**, all fault-free nodes are **enabled.**

Case 3. A five unit multiprocessor system, U_1 and U_4 are faulty units and enabled initially. Test results are **worst case**, disabling criteria is 2.

$\underline{U_4}$ U_3

U_0 0 0

U_0 $\underline{U_4}$

$\underline{U_1}$ 1 1

$\underline{U_1}$ U_0

U_2 1 0

U_2 $\underline{U_1}$

U_3 0 1

U_3 U_2

$\underline{U_4}$ 1 1

a. first iteration with I.C

all nodes are **enabled**

b. second iteration

U_0, U_1, U_2, U_3 are **enabled**

U_4 is **disabled**

* system stays in that state forever

* so system is not 2-fault 2-**correctable**

Case 4. This system is the same as case 3. Only the test results are arbitrary case.

$\underline{U_4}$ U_3

U_0 $\underline{0}$ 0

U_0 $\underline{U_4}$

$\underline{U_1}$ 1 $\underline{1}$

$\underline{U_1}$ U_0

U_2	$\underline{1}$	0
	U_2	$\underline{U_1}$
U_3	0	$\underline{1}$
	U_3	U_2
$\underline{U_4}$	1	1

a. first iteration with I.C

all nodes are **enabled**

b. second iteration

U_0, U_2, U_3 are **enabled**

U_1, U_4 are **disabled**

* all faulty nodes are **disabled**, all fault-free **enabled.**

Case 5. A seven unit multiprocessor system, with U_1, U_3, U_5 are faulty and enabled initially. Test results are **worst case** and disabling criteria is 1.

	U_6	$\underline{U_5}$	U_4
U_0	0	1	0
	U_0	U_6	$\underline{U_5}$
$\underline{U_1}$	1	1	0
	$\underline{U_1}$	U_0	U_6
U_2	1	0	0
	U_2	$\underline{U_1}$	U_0
$\underline{U_3}$	1	0	1
	$\underline{U_3}$	U_2	$\underline{U_1}$
U_4	1	0	1
	U_4	$\underline{U_3}$	U_2
$\underline{U_5}$	1	0	1

	U_5	U_4	U_3
U_6	1	0	1

a. first iteration with I.C

U_1, U_3, U_5 are **enabled**

U_0, U_2, U_4, U_6 are **disabled**

* system stays in that state forever. So system is not 3-fault 1-**correctable**

Case 6. This system is the same as previous case, but disabling criteia is 2.

a. first iteration with I.C

U_0, U_1, U_2, U_3, U_5 are **enabled**

U_4,U_6 are **disabled**

b. second iteration

U_0, U_1, U_2, U_5 are **enabled**

U_3, U_4, U_6 are **disabled**

c. third iteration

U_0, U_1, U_2, U_4, U_5, U_6 are **enabled**

U_3 **disabled**

d. fourth iteration

U_0, U_2, U_4, U_6 are **enabled**

U_1, U_3, U_5 are **disabled**

* all faulty nodes are **disabled,** all fault-free nodes are **enabled.**

Case 7. A seven unit multiprocessor system, U_1, U_3, U_5 are faulty units and enabled initially. Test results are **arbitrary case,** disabling criteria is 2.

	U_6	U_5	U_4
U_0	0	1	0

	U_0	U_6	U_5
U_1	1	1	1

	U_1	U_0	U_6
U_2	1	0	0

$$\begin{array}{cccc}
 & U_2 & \underline{U_1} & U_0 \\
\underline{U_3} & 1 & \underline{0} & 1 \\
 & \underline{U_3} & U_2 & \underline{U_1} \\
U_4 & \underline{1} & 0 & \underline{0} \\
 & U_4 & \underline{U_3} & U_2 \\
\underline{U_5} & 1 & \underline{0} & 1 \\
 & \underline{U_5} & U_4 & \underline{U_3} \\
U_6 & \underline{0} & 0 & \underline{1}
\end{array}$$

a. first iteration with I.C

all nodes are **enabled**

b. second iteration

U_0, U_2, U_4, U_6 are **enabled**

U_1, U_3, U_5 are **disabled**

* all faulty nodes are **disabled**, all fault-free are **enabled**

Case 8. A seven unit multiprocessor system, U_0, U_1, U_3, U_4 are faulty units and U_1, U_3, U_4 are enabled initially. Test results are **worst case**, disabling criteria is 1.

$$\begin{array}{cccc}
 & U_6 & U_5 & \underline{U_4} \\
\underline{U_0} & 1 & 1 & 0 \\
 & \underline{U_0} & U_6 & U_5 \\
\underline{U_1} & 0 & 1 & 1 \\
 & \underline{U_1} & \underline{U_0} & U_6 \\
U_2 & 1 & 1 & 0 \\
 & U_2 & \underline{U_1} & \underline{U_0} \\
\underline{U_3} & 1 & 0 & 0 \\
 & \underline{U_3} & U_2 & \underline{U_1} \\
\underline{U_4} & 0 & 1 & 0 \\
 & \underline{U_4} & \underline{U_3} & U_2
\end{array}$$

U_5	1	1	0
	U_5	$\underline{U_4}$	$\underline{U_3}$
U_6	0	1	1

a. first iteration with I.C

U_0, U_1, U_3, U_4 are **enabled**

U_2, U_5, U_6 are **disabled**

* system stays in that state forever. So it's not 4-fault 1-**correctable**.

Case 9. This case is the same as the previous case, except the test results are **arbitrary case**.

	U_6	U_5	$\underline{U_4}$
$\underline{U_0}$	1	1	$\underline{1}$
	$\underline{U_0}$	U_6	U_5
$\underline{U_1}$	0	1	1
	$\underline{U_1}$	$\underline{U_0}$	U_6
U_2	$\underline{1}$	$\underline{0}$	0
	U_2	$\underline{U_1}$	$\underline{U_0}$
$\underline{U_3}$	1	$\underline{1}$	$\underline{0}$
	$\underline{U_3}$	U_2	$\underline{U_1}$
$\underline{U_4}$	$\underline{1}$	1	$\underline{0}$
	$\underline{U_4}$	$\underline{U_3}$	U_2
U_5	$\underline{1}$	$\underline{0}$	0
	U_5	$\underline{U_4}$	$\underline{U_3}$
U_6	0	$\underline{1}$	0

a. first iteration with I.C

U_1 is **enabled**

U_0, U_2, U_3, U_4, U_5, U_6 are **disabled**

b. second iteration

U_0, U_1, U_4, U_5, U_6 are **enabled**

U_2, U_3 are **disabled**

c. third iteration

U_4 is **enabled**

U_0, U_1, U_2, U_3, U_5, U_6 are **disabled**

d. fourth iteration

U_1, U_2, U_3, U_4 are **enabled**

U_0, U_5, U_6 are **disabled**

e. fifth iteration

U_1 is **enabled** and U_0, U_2, U_3, U_4, U_5, U_6 are **disabled.**

* This is iteration #1.So system is not 4-fault, 1- **correctable.**

Case 10. This system is the same as case 8, disabling criteria is 2 in this case. The test results will be the same as in case #8.

a. first iteration with I.C

U_0, U_1, U_2, U_3, U_4 are **enabled**

U_5, U_6 are **disabled**.

b. second iteration

U_0, U_1, U_3, U_4 are **enabled**

U_2, U_5, U_6 are **disabled**

c. third iteration

U_0, U_1, U_3, U_4 are **enabled**

U_2, U_5, U_6 are **disabled**

* This is I.C (initial condition) state, system stays in that loop for ever. That means system is not 4-fault 2-**correctable.**

Case 11. This is the same as case 9, disabling criteia is 2 in this case. Test results will be the same as in case #9.

a. first iteration

all nodes are **enabled**

b. second iteration

U_2, U_5, U_6 are **enabled**

U_0, U_1, U_3, U_4 are **disabled**

* all faulty units are **disabled**, all fault-free units are **enabled**.

Case 12. An eight unit multiprocessor system, U_0, U_3, U_5, U_7 are faulty units and U_0, U_3, U_5 are enabled initially. Test results are **worst case**, disabling criteria is 1.

	$\underline{U_7}$	U_6	$\underline{U_5}$
$\underline{U_0}$	0	1	0
	$\underline{U_0}$	$\underline{U_7}$	U_6
U_1	1	1	0
	U_1	$\underline{U_0}$	$\underline{U_7}$
U_2	0	1	1
	U_2	U_1	U_8
$\underline{U_3}$	1	1	0
	$\underline{U_3}$	U_2	U_1
U_4	1	0	0
	U_4	$\underline{U_3}$	U_2
$\underline{U_5}$	1	0	1
	$\underline{U_5}$	U_4	$\underline{U_3}$
U_6	1	0	1
	U_6	$\underline{U_5}$	U_4
$\underline{U_7}$	1	0	1

a. first iteration with I.C

U_0, U_3, U_5, U_7 are **enabled**

U_2, U_1, U_4, U_6 are **disabled**

* system stays in that forever. So system is not 4-fault, 1-**correctable**.

When we try to simulate if the system is 2-correctable.

We can easily see that U_0 will never be disabled in that case. So system is not 2-**correctable** either.

Case 13. This the same as case 12, but test results are **arbitrary** and disabling criteria is 2.

	$\underline{U_7}$	U_6	$\underline{U_5}$
$\underline{U_0}$	$\underline{0}$	1	$\underline{1}$
	$\underline{U_0}$	$\underline{U_7}$	U_6
U_1	$\underline{1}$	$\underline{0}$	0
	U_1	$\underline{U_0}$	$\underline{U_7}$
U_2	0	1	0
	U_2	U_1	$\underline{U_0}$
$\underline{U_3}$	1	1	$\underline{0}$
	$\underline{U_3}$	U_2	U_1
U_4	$\underline{1}$	0	0
	U_4	$\underline{U_3}$	U_2
$\underline{U_5}$	1	$\underline{1}$	1
	$\underline{U_5}$	U_4	$\underline{U_3}$
U_6	$\underline{0}$	0	$\underline{1}$
	U_6	$\underline{U_5}$	U_4
$\underline{U_7}$	1	$\underline{0}$	1

a. first iteration with I.C

all nodes are **enabled.**

b. second iteration

U_1, U_2, U_4 and U_6 are **enabled**

U_0, U_3, U_5, U_7 are **disabled**

* all faulty units are **disabled,** all fault-free units are **enabled**.

Case 14. A nine unit multiprocessor system, U_0, U_1, U_2, U_3 are faulty units and U_0, U_2, U_3 are enabled. Test results are **worst case**, disabling criteria is 1.

	U_8	U_7	U_6	U_5
$\underline{U_0}$	1	1	1	1
	$\underline{U_0}$	U_8	U_7	U_6
$\underline{U_1}$	0	1	1	1
	$\underline{U_1}$	$\underline{U_0}$	U_8	U_7
$\underline{U_2}$	0	0	1	1
	$\underline{U_2}$	$\underline{U_1}$	$\underline{U_0}$	U_8
$\underline{U_3}$	0	0	0	1
	$\underline{U_3}$	$\underline{U_2}$	$\underline{U_1}$	$\underline{U_0}$
U_4	1	1	1	1
	U_4	$\underline{U_3}$	$\underline{U_2}$	$\underline{U_1}$
U_5	0	1	1	1
	U_5	U_4	$\underline{U_3}$	$\underline{U_2}$
U_6	0	0	1	1
	U_6	U_5	U_4	$\underline{U_3}$
U_7	0	0	0	1
	U_7	U_6	U_5	U_4
U_8	0	0	0	0

a. first iteration with I.C

U_0, U_1, U_2, U_3, U_8 are **enabled**

U_4, U_5, U_6, U_7 are **disabled**

b. second iteration

U_8 is **enabled**

all the others are **disabled**

c. third iteration

U_4, U_5, U_6, U_7, U_8 are **enabled**

U_0, U_1, U_2, U_3 are **disabled**

* all faulty nodes are **disabled**, fault-free nodes are **enabled.**

Case 15. A nine unit multiprocessor system,U_0, U_3, U_5, U_8 are faulty units and U_3, U_5, U_8 are enabled. Test results are **arbitrary case**, disabling criteria is 1.

	$\underline{U_8}$	U_7	U_6	$\underline{U_5}$
$\underline{U_0}$	$\underline{1}$	1	1	$\underline{0}$
	$\underline{U_0}$	$\underline{U_8}$	U_7	U_6
U_1	$\underline{1}$	$\underline{0}$	0	0
	U_1	$\underline{U_0}$	$\underline{U_8}$	U_7
U_2	0	$\underline{1}$	$\underline{1}$	0
	U_2	U_1	$\underline{U_0}$	$\underline{U_8}$
$\underline{U_3}$	1	1	$\underline{0}$	$\underline{1}$
	$\underline{U_3}$	U_2	U_1	$\underline{U_0}$
U_4	$\underline{1}$	0	0	$\underline{0}$
	U_4	$\underline{U_3}$	U_2	U_1
$\underline{U_5}$	1	$\underline{0}$	1	1
	$\underline{U_5}$	U_4	$\underline{U_3}$	U_2
U_6	$\underline{1}$	0	$\underline{0}$	0
	U_6	$\underline{U_5}$	U_4	$\underline{U_3}$
U_7	0	$\underline{1}$	0	$\underline{1}$
	U_7	U_6	$\underline{U_5}$	U_4
$\underline{U_8}$	1	1	$\underline{0}$	1

a. first iteration with I.C

U_1, U_5, U_8 are **enabled**

U_0, U_2, U_3, U_4, U_6, U_7 are **disabled**

b. second iteration

U_1, U_4, U_8 are **enabled**

U_0, U_2, U_3, U_5, U_6, U_7 are **disabled**

c. third iteration

U_1, U_4, U_6, U_7 are **enabled**

U_0, U_2, U_3, U_5, U_8 are **disabled**

d. fourth iteration

U_1, U_2, U_4, U_6, U_7 are **enabled**

U_0, U_3, U_5, U_8 are **disabled**

* All faulty nodes are **disabled**, all fault-free nodes are **enabled.**

Case 16. This the same as previous case but disabling criteria is 2.

a. first iteration

U_0, U_1, U_2, U_3, U_4, U_5, U_6, U_8 are **enabled**

U_7 is **disabled**

b. second iteration

U_1, U_4, U_6 are **enabled**

U_0, U_2, U_3, U_5, U_7, U_8 are **disabled**

c. third iteration

U_0, U_1, U_2, U_3, U_4, U_6, U_7 are **enabled**

U_5, U_8 are **disabled**

d. fourth iteration

U_1, U_2, U_4, U_6, U_7 are **enabled**

U_0, U_3, U_5, U_8 are **disabled.**

* All faulty nodes are **disabled**, all fault-free nodes are **enabled.**

Case 17. This case is the same as case 15, but disabling criteria is 3.

a. first iteration

all nodes will be **enabled**

b. second iteration

U_1, U_2, U_4, U_6, U_7 are **enabled**

U0, U_3, U_5, U_8 are **disabled**

* All faulty nodes are **disabled**, all fault-free nodes are **enabled.**

Case 18. A nine unit multiprocessor system, U_1, U_3, U_5, U_8 are faulty units and U_1, U_3, U_5 are enabled. Test results **are worst case,** disabling criteria is 2.

	$\underline{U_8}$	U_7	U_6	$\underline{U_5}$
U_0	1	0	0	1
	U_0	$\underline{U_8}$	U_7	U_6
$\underline{U_1}$	1	0	1	1
	$\underline{U_1}$	U_0	$\underline{U_8}$	U_7
U_2	1	0	1	0
	U_2	$\underline{U_1}$	U_0	$\underline{U_8}$
$\underline{U_3}$	1	0	1	0
	$\underline{U_3}$	U_2	$\underline{U_1}$	U_0
U_4	1	0	1	0
	U_4	$U_{\underline{3}}$	$U_{\underline{2}}$	$U_{\underline{1}}$
$\underline{U_5}$	1	0	1	0_
	$\underline{U_5}$	U_4	$\underline{U_3}$	U_2
U_6	1	0	1	0
	U_6	$U_{\underline{5}}$	$U_{\underline{4\ }}$	$\underline{U_3}$
U_7	0	1	0	1
	U_7	U_6	$\underline{U_5}$	U_4
$\underline{U_8}$	1	1	0	1

a. first iteration with I.C

$U_0, U_1, U_2, U_3, U_5, U_8$ are **enabled**

U_4, U_6, U_7 are **disabled**.

b. second iteration

U_1, U_5, U_8 are **enabled**

$U_0, U_2, U_3, U_4, U_6, U_7$ are **disabled.**

c. third iteration

$U_1, U_3, U_4, U_5, U_6, U_7, U_8$ are **enabled**

U_0, U_2 are **disabled.**

d. fourth iteration

U_3, U_5 are **enabled**.

$U_0, U_1, U_2, U_4, U_6, U_7, U_8$ are **disabled**.

e. fifth iteration

$U_0, U_1, U_2, U_3, U_4, U_5, U_8$ are **enabled**

U_6, U_7 are **disabled.**

f. sixth iteration

U_1, U_8 are **enabled**

$U_0, U_2, U_3, U_4, U_5, U_6, U_7$ are **disabled**.

* System is not 4-fault 2-**correctable.**

Case 19. A six unit multiprocessor system, U_1, U_3, U_5 are faulty units and only U_1 is disabled, all the other units are enabled. Test results are **arbitrary case**, disabling criteria is 2.

	$\underline{U_5}$	U_4
U_0	$\underline{0}$	0

	U_0	$\underline{U_5}$
$\underline{U_1}$	1	1

	$\underline{U_1}$	U_0

U_2 $\underline{0}$ 0

U_2 $\underline{U_1}$

$\underline{U_3}$ 1 $\underline{1}$

$\underline{U_3}$ U_2

U_4 $\underline{0}$ 0

U_4 $\underline{U_3}$

$\underline{U_5}$ 1 1

a. first iteration

U_0, U_2, U_3, U_4 are **enabled**

U_1, U_5 are **disabled**.

b. second iteration

U_0, U_1, U_2, U_3, U_4 are **enabled**.

U_5 is **disabled**.

c. third iteration

U_0, U_1, U_2, U_4 are **enabled.**

U_3, U_5 are **disabled**.

d. fourth iteration

U_0, U_1, U_2, U_4, U_5 are **enabled**.

U_3 is **disabled**.

e. fifth iteration

U_0, U_2, U_4, U_5 are **enabled.**

U_1, U_3 are **disabled**.

f. sixth iteration

U_0, U_2, U_3, U_4, U_5 are **enabled**.

U_1 is **disabled.**

*That is I.C state and system oscillates and returns to I.C state in every six iteration.

LIST OF REFERENCES

1. J.H. Wesley, et. al., "SIFT: Design and analysis of fault tolerant computer for Aircraft Control," Proc. of IEEE,Vol. 66, No. 10, pp. 1240-1255, October 1978.
2. F.P. Preparata, G.Metze, and R.T.Chien.,"On the connection assignment problem of diagnosable systems," IEEE Trans. on Comp., Vol. C-16, pp. 848-854, Dec.1967.
3. K.Y. Chwa and S.L.Hakimi, "Schemes for fault tolerant computing: A comparison of modularly redundant and t-diagnosable systems," Inform. and Control, Vol. 49, No. 3, pp. 212-238, June 1981.
4. Arthur D.Friedman and Luca Simoncini, "System level fault diagnosis," IEEE Trans. on Comp., Vol. 13, p. 47-2, March 1980.
5. Simoncini Karunanithi and A.D. Friedman,"System diagnosis with t/s diagnosability," Proc. of the 7 th Fault-tolerant Comp. Symp., pp. 65-71, June 1977
6. M.L. Blount, "Probabilistic Treatment of Diagnosis in Digital systems, Proc. 7th Intl. Conf. on Fault Tolerant Computing, pp. 72-77, June 1977.
7. J.T. Butler, "On the design of distributed diagnosable multiprocessing systems," Naval Postgraduate School Monterey, CA, research proposal.
8. A.L. Hopkins, T.B. Smith, and J.H. Lala, " FTMP-A highly reliable fault-tolerant multiprocessor for aircraft," Proc. of IEEE, Vol. 66, No. 10, pp. 1221-1239, October 1978.
9. R. Nair, G. Metze and J. Abraham, "Design Considerations for Fault -Tolerant Distributed Digital Systems," unpublished manuscript.
10. S. Mallela and G. Masson, "Diagnosable systems for intermittent faults," IEEE Trans. on Comp., Vol. C-27, pp. 560-566, 1978.
11. Stephan G. Kochan "Programming in C," Hayden Book Company, 1983.

INITIAL DISTRIBUTION LIST

No.copies

1. Defense Technical Information Center 2
Cameron Station
Alexandra, VA 22304-6145

2. Library, Code 0142 2
Naval Postgraduate School
Monterey, CA 93943-5002

3. Department Chairman, Code 62 1
Department of Electrical and Computer Engineering.
Naval Postgraduate School
Monterey, CA 93943-5000

4. Dr. Jon T. Butler, Code 62 BU 5
Department of Elecrical and ComputerEngineering.
Naval Postgraduate School
Monterey, CA 93943-5000

5. Dr. Bruno O. Shubert, Code 55 SY 1
Department of Operational Analysis
Naval Postgraduate School
Monterey, CA 93943

6. Dr. Dana E. Madison, Code 52 1
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943

7. Dr. Joo Kang Lee 1
POSTECH Research Institute of Science
and Technology
PO.Box.125, Pohang City
Kyungbuk 680 KOREA.

8. Director of Research Administration, Code 012 1
Naval Postgraduate School
Monterey, CA 93943-5000

9. Kara Kuvvetleri Komutanligi 1
Egitim Dairesi Baskanligi
Bakanliklar, Ankara, Turkey

10. Kara Harp Okulu 1
Bakanliklar, Ankara, Turkey

11. Muhabere Okul Komutanligi 1
Mamak, Ankara, Turkey

12. Capt. Ibrahim Dincer 1
Muhaber okulu
Ogretim Kurulu
Mamak, Ankara, Turkey

13. Ltjg. Mustafa Paktuna 1
Marmara cad. No:158/6
Kocamustafapasa, Istanbul, Turkey

14. Dr. Andre von Tibborg 1
Code 1133
ONR
800 N. Quincy
Arlington, VA 22217

15. Dr. George Abraham 1
Code 7500
NRL
4555 Overlook Ave. S.W.
Washington, DC 20375

16. Dr. Lou Schmid 1
ONT 20T
800 N. Quincy Ave. Room 811
Arlington, VA 22217